

Dokumentation Arduino Pager

Autoren: Thilo Drehlmann, Gerrit Koppe

Ausbildungsberuf: Fachinformatiker für Anwendungsentwicklung

Lernfeld 7: Cyberphysische Systeme ergänzen

21. Januar 2023

Inhaltsverzeichnis

1	Einleitung	1
2	Ausgangssituation / Formulierung des Themas	1
2.1	Beschreibung des Projektes	1
2.2	Definition der Ziele, erwartetes Ergebnis	1
3	Ressourcen und Ablaufplanung	1
3.1	Allgemeine und Ressourcenplanung	1
3.2	Ressourcen	2
3.2.1	Benötigte Hardware	2
3.2.2	Benötigte Software	2
3.2.3	Zeitliche Planung der Umsetzung	2
3.2.4	Erwartete Schwierigkeiten	3
3.2.5	Arbeitsaufteilung	3
4	Durchführung	3
4.1	Schaltplan und Bau des Projekts	3
4.2	Konzeptionierung der Quellcode-Architektur	4
4.3	Umsetzung des Quellcodes	4
5	Projektergebnis	6
5.1	Qualitätskontrolle	6
5.2	Soll-Ist-Vergleich	7
5.3	Ausblick	7
6	Anlagen	8
6.1	Auflistung der Komponenten	8
6.2	Diagramme	9
6.3	Bilder	18
6.4	Quellcode Auszüge	22
6.4.1	Klasse: Messenger	22
6.4.2	Klasse: Radio	24
6.4.3	Klasse: VKeys	27
6.4.4	Klasse: ColorChooser	28
7	Glossar	29
7.1	Technische Begriffe	29
7.2	Arduino Begriffe	29
8	Quellenverzeichnis	30
8.1	Internetquellen	30
8.2	Links genutzter Hardware	30
8.3	Links genutzter Software	30

1 Einleitung

In dieser Dokumentation wird die Umsetzung eines bidirektionalen Pagers auf Basis der Arduino Plattform beschrieben. Zunächst werden Thema und Ziel des Projekts formuliert. Anschließend wird auf die Planung der Ressourcen und des Ablaufs, sowie auf die benötigten Komponenten eingegangen. Im Anschluss wird das Vorgehen während des Projektes dokumentiert und abschließend das Ergebnis der Durchführung präsentiert.

2 Ausgangssituation / Formulierung des Themas

2.1 Beschreibung des Projektes

Das Thema des Projektes ist es, eine bidirektionale Kommunikation zwischen zwei Geräten auf Arduino-Basis zu gewährleisten. Es soll die Möglichkeit bestehen, Nachrichten zu verfassen, zu versenden und ebenso Nachrichten zu empfangen, die von einem anderen Arduino Gerät versendet wurden.

2.2 Definition der Ziele, erwartetes Ergebnis

Im Folgenden werden die allgemein Ziele des Projektes näher definiert.

1. Es soll möglich sein, mittels eines Touchscreens und einer virtuellen Tastatur, Zeichenketten auf einem, an den Arduino angeschlossenen Touchscreen, zu schreiben.
 2. Die eingegebenen Zeichenketten sollen, mittels Funkwellen, an ein anderes Gerät übertragen werden können.
 3. Das Gerät soll in der Lage sein, Funkwellen zu empfangen.
 4. Das Gerät soll außerdem in der Lage sein, die empfangenen Funkwellen wieder zu einer Zeichenkette zu übersetzen und auf einem Touchscreen anzuzeigen.
 5. Es soll ein graphisches User Interface auf dem Touchscreen geben.
 6. Es soll möglich sein, empfangene Nachrichten zwischenspeichern, damit neu empfangene Nachrichten nicht die vorherigen Nachrichten überschreiben.
 7. Es soll möglich sein, den Zwischenspeicher der Nachrichten über einen eigenen Menüpunkt abzurufen und die empfangenen Nachrichten zu verwalten.
- Außerdem gibt es folgende, optionale Ziele:
- a. Das User Interface soll anpassbar sein (Farben, Tastaturlayout).
 - b. Es soll möglich sein, zu überprüfen, ob empfangsbereite Geräte in der Nähe sind.
 - c. Es soll möglich sein, die Antenne zu konfigurieren (Distanz und Frequenz).

3 Ressourcen und Ablaufplanung

3.1 Allgemeine und Ressourcenplanung

Das Projekt besteht aus zwei Teilen, der drahtlosen Kommunikation zweier Arduino Geräte miteinander und dem User Interface / der menschlichen Schnittstelle mit der Hardware, über die Nachrichten angezeigt, aber auch eingegeben und versendet werden können. Für die Planung ist also zunächst relevant, welche Hardware Komponenten benötigt werden, um beide Projektteile umzusetzen.

Für die drahtlose Kommunikation kann auf verschiedene Technologien wie Bluetooth oder Radiofrequenz zurückgegriffen werden. Da Bluetooth nur eine sehr eingeschränkte Reichweite besitzt und eine Kopplung der Geräte notwendig ist, wurde sich dafür entschieden, die drahtlose Übertragung per Radiofrequenz durchzuführen. Außerdem existieren für eine solche Übertragung kostengünstige Hardware und fertige API-Bibliotheken für die Hardware. Aus diesen Gründen wurde sich für die nRF24L01+ Transceiver (im Folgenden als „Antenne“ oder „Transceiver“ bezeichnet) entschieden.¹

¹Vgl. Hardware Link 1

Für die Umsetzung der menschlichen Schnittstelle sollten zunächst ein Numpad, sowie ein 16x2 Liquid Crystal Display verwendet werden. Aufgrund der stark eingeschränkten Nutzbarkeit des LCD, sowie umständlicher Eingabe über das Numpad, wurde sich stattdessen entschieden, einen Touchscreen zu verwenden. Da bereits Arduino Uno R3 Boards der Marke Elegoo zur Verfügung standen, wurde sich hier für den Elegoo TFT 2,8" Touchscreen (im Folgenden als „TFT“ oder „Touchscreen“ bezeichnet) entschieden.² Dieser hat außerdem den Vorteil, dass Elegoo Bibliotheken mitliefert, um den Touchscreen zu programmieren.

Schlussendlich musste noch entschieden werden, welches Arduino Board verwendet werden sollte. Aufgrund guter Erfahrung mit der Marke Elegoo in Hinsicht auf Zuverlässigkeit und Nutzbarkeit, wurde sich für ebenjene Marke entschieden. Da der Touchscreen als Shield³ für Arduino Uno Boards konzipiert ist, wäre es bei einem Arduino Uno nicht mehr möglich, weitere Hardware Komponenten anzuschließen. Entsprechend fiel die Entscheidung auf den Elegoo Arduino Mega R3⁴, da dieser deutlich mehr Pinouts zur Verfügung stellt, als ein Arduino Uno. Weitere Gründe für diese Entscheidung sind, dass der Transceiver über den SPI Bus des Arduinos kommuniziert, dieser wäre bei einem Arduino Uno aber durch den TFT blockiert. Außerdem wird viel Quellcode erwartet, was bedeutet, dass der Programmerspeicher des Arduino Uno zu klein sein könnte, auch hier hat der Arduino Mega mehr Ressourcen.

3.2 Ressourcen

3.2.1 Benötigte Hardware

In Anhang Tabelle 1 findet sich eine detaillierte, tabellarische Auflistung aller in Kapitel 3.1 genannten Komponenten, ihrer Aufgaben und ihrer Preise. Alle Komponenten werden zwei mal benötigt, da eine Kommunikation zwischen zwei identischen Geräten hergestellt werden soll.

3.2.2 Benötigte Software

Zur Umsetzung des Projekts wird, um die Programmierung zu vereinfachen und den Quellcode schlanker zu halten, auf verschiedene externe Bibliotheken zurückgegriffen. Eine detaillierte Auflistung dieser Bibliotheken findet sich im Anhang Tabelle 2.

3.2.3 Zeitliche Planung der Umsetzung

Ein Netzplan mit detaillierter Zeitplanung findet sich im Kapitel 6.2 Abbildung 2. Der Plan lässt sich weiterhin unterteilen in die Planungsphase (Schritte 1-6) und die Umsetzungsphase (alle weiteren Schritte), abgebildet in Kapitel 6.2 Abbildung 3 und Abbildung 4.

Da vor der Umsetzung des Projekts keine Erfahrung im Umgang mit der Arduino Plattform bestand, wurden bei der Planung der gesamten Planungsphase (Netzplan Schritte 1-6) viel Zeit eingeräumt, durch die fehlendes Wissen und Einarbeitung kompensiert werden sollten. Des Weiteren wurden 6, respektive 8 Stunden geplant, damit die Umsetzenden sich mit den einzelnen Komponenten vertraut machen konnten (Netzplan Schritte 4 und 5). Nach erfolgreicher Einarbeitung wurden weitere 2 Stunden für die Konzeptionierung des Quellcodes eingeplant⁵. Die zeitliche Planung der Umsetzung wurde erst nach der Planungsphase durchgeführt, da diese abhängig vom, in der Planung entstandenen, Design des Quellcodes war.⁶

Aufgrund des entstandenen Quellcode Designs konnte die Entwicklung der einzelnen Klassen parallel ablaufen und nur die Klasse „Messenger“ war davon abhängig, dass die anderen Klassen bereits fertig gestellt waren. Entsprechend wurde die Entwicklung der Klassen „VKeys“, „Radio“ und „ColorChooser“ zeitgleich eingeplant, wobei der Radio-Klasse die längste Dauer der Entwicklung eingeplant

²Vgl. Hardware Link 3

³Vgl. Glossar 7.2: Shield

⁴Vgl. Hardware Link 2

⁵Ergebnis abgebildet in Kapitel 6.2 Abbildung 6.

⁶Alle bis hier beschriebenen Schritte sind in Kapitel 6.2 Abbildung 3 abgebildet

wurde. Jeder Klasse wurde außerdem eine Testphase, entsprechend ihrer erwarteten Komplexität, eingeplant.

Im Anschluss wurde die Messenger-Klasse mit dem größten zeitlichen Aufwand von 14 Stunden Entwicklung und 5 Stunden Testen eingeplant, da diese die Kommunikation aller Komponenten sicherstellen soll.

Abschließend wurden noch 5 Stunden für Tests des fertigen Systems und abschließendes Debuggen eingeplant, sowie das Schreiben dieser Dokumentation.⁷

3.2.4 Erwartete Schwierigkeiten

Im Folgenden werden alle Schwierigkeiten aufgelistet und erklärt, die während der Umsetzung des Projekts erwartet werden.

1. *Fehlersuche bei fehlerhafter Übertragung*: Da dieses Projekt darauf basiert, Funksignale zu versenden und zu empfangen und die Umsetzenden keine Gerätschaft besitzen, Funkwellen und Signalstärken dieser zu messen, wird es schwierig, den Fehler zu identifizieren, sollte eine Übertragung fehlschlagen.

2. *Distanzregulierung*: Die Transceiver können in verschiedenen Signalstärken senden, die programmatisch eingestellt werden müssen. Wird eine zu hohe Signalstärke konfiguriert, leidet darunter allerdings die Übertragungsqualität bei niedrigen Distanzen. Hier muss ein gutes Mittelmaß gefunden werden.

3. *Wechsel zwischen Empfang und Senden*: Da die Nachrichten bidirektional versendet werden sollen, die Transceiver aber nur halbduplex arbeiten, muss ein rechtzeitiger Wechsel der Antenne zwischen Senden und Empfang garantiert werden. Sollten beide Geräte gleichzeitig Senden, werden beide Nachrichten verloren gehen.

4. *Kein Multithreading*: Da Arduinos nicht Multithreading-fähig⁸ sind, wird eine Schwierigkeit darin bestehen, permanent gleichzeitig die Nutzung des Geräts und den Empfang eingehender Nachrichten zu gewährleisten.

3.2.5 Arbeitsaufteilung

Die genaue Aufteilung der Aufgaben ist dem Netzplan in Abbildung 2 zu entnehmen.

Die Projektschritte wurden allgemein alle gemeinsam unternommen, da die Hardware doppelt aufgebaut werden musste und der Quellcode in einem Git Repository verwaltet wurde, wodurch die Umsetzenden parallel an den gleichen Klassen arbeiten konnten. Um Zeit zu sparen, wurde nur bei Arbeitsschritten, die parallel ablaufen konnten (z.B. Schritte 7-12, vgl. Abbildung 4) die Arbeit getrennt voneinander durchgeführt.

4 Durchführung

Im Folgenden wird die Umsetzung des Projekts beschrieben. Alle im Text benannten Methoden finden sich in Kapitel 6.4 unter dem jeweiligen Subkapitel der Klasse, zu der die Methode gehört.

4.1 Schaltplan und Bau des Projekts

Zunächst wurde die Hardware zusammengebaut. Da der Touchscreen als Shield entworfen ist, konnte dieser einfach auf den Arduino Mega gesteckt werden.

Die Transceiver benötigten wiederum eine 5V Stromversorgung, sowie den SPI-Bus⁹ des Arduinos. Der SPI-Bus des Arduino Mega 2560 R3 besteht aus den Pins D50-D53¹⁰, welche nicht durch den

⁷Alle bis hier beschriebenen Schritte sind in Kapitel 6.2 Abbildung 4 abgebildet

⁸vgl. Glossar 7.1: Multithreading

⁹Vgl. Glossar 7.1: SPI

¹⁰Vgl. Abbildung 11

Touchscreen-Shield verwendet werden, weshalb hier keine Probleme entstehen. Für die 5V Stromversorgung wird einer der beiden 5V-Pins in der Pin-Reihe D22-D53 verwendet, da diese noch nicht durch den Touchscreen-Shield verwendet werden.

Somit kann das Projekt ohne Benutzung weiterer Komponenten gebaut werden. Ein Bild des aufgebauten Projekts befindet sich in Kapitel 6.3, Abbildung 14.

Der Schaltplan des Projekts ist in Abbildung 5 dargestellt.

4.2 Konzeptionierung der Quellcode-Architektur

Nachdem das Projekt erfolgreich aufgebaut wurde, musste der Quellcode geschrieben werden. Da die Stromkreise der beiden Komponenten komplett getrennt sind, musste die Kommunikation dieser per Quellcode erfolgen. Zunächst wurden die Funktionalitäten definiert, die das Projekt bieten muss. Diese wurden anschließend thematisch aufgeteilt und einzelnen Klassen zugeordnet. Die benötigten Funktionalitäten wurden direkt aus den Zielen in Kapitel 2.2 abgeleitet. Die entsprechenden Ziele werden im Folgenden durch Fußnoten referenziert. Ziele können durch mehrere Klassen realisiert werden und können eventuell mehrfach aufgeführt werden.

So wurden alle Funktionalitäten, welche mit dem Versenden und Empfangen von Nachrichten zu tun haben, in einer Klasse gekapselt, welche „Radio“ genannt wurde. Diese Klasse sollte sowohl für das Versenden und Empfangen von, durch den User eingegebenen, Nachrichten, als auch für die Überprüfung auf Geräte in der Nähe und die Implementierung der RF24 API verantwortlich sein.¹¹

Eine weitere Aufgabe sollte die Implementierung einer Möglichkeit zur Eingabe von Nachrichten sein. Hier wurde sich dafür entschieden, eine virtuelle Tastatur auf dem Touchscreen anzuzeigen. Die Anzeige dieser Tastatur, sowie das Erkennen des eingegebenen Buchstaben, sollten in der Klasse „VKeys“ gekapselt werden.¹²

Des weiteren sollte eine eigene Klasse zur Auswahl von Farben erstellt werden, welche immer dann instanziiert werden sollte, wenn der User einen Menüpunkt auswählt, mit dem das Aussehen des User Interface verändert werden kann. Diese Klasse wurde „ColorChooser“ benannt und sollte alle möglichen Farben anzeigen und erkennen, welche Farbe ausgewählt wurde.¹³

Letztendlich musste noch eine Klasse konzeptioniert werden, die für das User Interface verantwortlich sein sollte. Außerdem wurde sich entschieden, diese Klasse zu verwenden, um alle anderen Klassen zu implementieren und die gesamte Logik zentral zu steuern. Diese Klasse wurde „Messenger“ benannt. Diese Klasse sollte außerdem für die Menüführung und das Zwischenspeichern von Nachrichten verwendet werden.¹⁴

Aus der Konzeptionierung ergab sich dann ein Klassendiagramm, welches in vereinfachter Version in Abbildung 6 dargestellt ist.

4.3 Umsetzung des Quellcodes

Im Folgenden werden die Implementierung der einzelnen Funktionalitäten, sowie die dabei aufgetretenen Schwierigkeiten beschrieben. Wird bei der Beschreibung der Umsetzung ein Netzplan referenziert, so ist der Netzplan in Abbildung 4 gemeint.

Da die konzeptionierte Architektur vorsah, eine Hauptklasse zu entwerfen, die wiederum 3 weitere Klassen verwendet, wurde sich entschieden, bei der Entwicklung des Quellcodes den Bottom-Up Ansatz¹⁵ zu verwenden und zunächst die 3 verwendeten Klassen („VKeys“, „ColorChooser“ und „Radio“) zu entwickeln. Wie im Netzplan zu erkennen, wurden diese parallel entwickelt.

Sowohl bei der Entwicklung von „ColorChooser“, als auch bei der von „VKeys“ ist aufgefallen, dass Touchscreen und Display, obwohl es die selbe physische Komponente ist, programmatisch zwei verschiedene Koordinatensysteme haben. Somit musste zunächst durch Ausprobieren herausgefunden

¹¹Vgl. Kapitel 2.2, Ziele: 2,3,4,b,c

¹²Vgl. Kapitel 2.2, Ziele: 1,a

¹³Vgl. Kapitel 2.2, Ziel: a

¹⁴Vgl. Kapitel 2.2, Ziele: 2,5,6,7,c

¹⁵Vgl. Glossar 7.1: Bottom-Up

werden, dass das Display (x,y)-Koordinaten im Bereich $([0, 240], [0, 320])$ und der Touchscreen (x,y)-Koordinaten im Bereich $([120, 940], [70, 920])$ besitzt. Noch dazu ist die Y-Achse der beiden Komponenten zueinander invertiert (Display-Y: 0, Touchscreen-Y: 920 und Display-Y: 320, Touchscreen-Y: 70). Dieser Sachverhalt ist in Kapitel 6.2, Abbildung 12 graphisch dargestellt. Um diese Diskrepanz zu überbrücken, musste eine Methode zum Übersetzen der Koordinaten von Touchscreen zu Display geschrieben werden, um User Eingaben trotzdem korrekt zu verarbeiten.¹⁶

Abgesehen von der genannten Schwierigkeit, konnten beide Klassen aber im geplanten Zeitrahmen entwickelt werden. Im Weiteren wird hier aber hauptsächlich die VKeys Klasse beschrieben, da „ColorChooser“ hauptsächlich entwickelt wurde, um sich in einer kleinen Klasse mit dem TouchScreen vertraut zu machen.

Genutzt wurde bei der Entwicklung von „VKeys“ hauptsächlich die von Elegoo mitgelieferte Schnittstelle zum Display, die in den Bibliotheken „TouchScreen“ und „Elegoo_TFTLCD“ gekapselt wurde¹⁷. Für beide Klassen wurde jeweils eine Methode geschrieben, um die TouchScreen-Berührung zu interpretieren und, anhand der eingegebenen Koordinaten, den Wert der gewählten Taste zurückzugeben¹⁸ (z.B.: Wenn bei der virtuellen Tastatur der Touchscreen an der Stelle berührt wird, an der das „A“ gedruckt ist, gibt die Methode auch „A“ zurück). Des Weiteren wurde für beide Klassen eine Methode geschrieben, die ihr jeweiliges Interface auf den Bildschirm druckt.¹⁹

Ebenfalls parallel wurde die Klasse „Radio“ entwickelt, die alle Funktionalitäten zur Kommunikation der beiden Geräte beinhalten sollte. Um die Transceiver ansteuern zu können, wurde die RF24 Treiber Bibliothek von TMRh20 verwendet²⁰. Hier wurden zunächst die Methoden zum Senden²¹ und zum Empfangen²² der Nachrichten entwickelt. Beim Empfangen der Nachrichten musste darauf geachtet werden, dass diese Methode später auch genutzt werden sollte, um Kommandos zu empfangen, wie zum Beispiel die Anfrage auf Verfügbarkeit, welche wiederum nicht die empfangene Nachricht zurückgeben sollte, sondern die Verfügbarkeit zurückmelden.

Eine weitere Schwierigkeit bestand in der Typisierung der zu übertragenden Nachrichten. Es wurde, der Übersichtlichkeit und Einfachheit halber, die meiste Zeit mit dem Datentyp String gearbeitet, um Zeichenketten zwischen Methoden zu übergeben. Da dieser Datentyp aber keine Informationen beinhaltet, sondern nur einen Pointer zu einem char-Array, kann ein String nicht übertragen werden. Um dieses Problem zu umgehen, musste der String vor der Übertragung zu einem char-Array konvertiert werden und nach dem Empfangen wieder zurück zu einem String. Hierfür wurden in der Radio-Klasse Methoden geschrieben, die automatisch aufgerufen werden, sobald eine Nachricht versendet oder empfangen wird²³.

Um die erwartete Schwierigkeit 3 in Kapitel 3.2.4 präventiv zu unterbinden, wurde sich entschieden, die Klasse so zu bauen, dass beim Verlassen jeder Methode die Antenne wieder auf lauschen gestellt wird und das Senden nur kurzzeitig aktiviert wird, sollte eine Nachricht übertragen werden.

Schlussendlich wurde für die „Radio“ Klasse noch die Methode geschrieben, um auf verfügbare Geräte zu überprüfen. Diese sollte eine vordefinierte Kommandosequenz senden und anschließend abwarten, ob eine weitere vordefinierte Kommandosequenz zurückgesendet wird²⁴.

Alle bis zu diesem Punkt entwickelten Klassen wurden einzeln getestet.

Zuletzt wurde die Klasse „Messenger“ entwickelt, die alle Funktionalitäten des Projekts kapseln und die bisher entwickelten Klassen implementieren sollte²⁵.

Zunächst wurden der allgemeine Aufbau des User Interface, sowie die Logik der Menüführung definiert. Es wurde entschieden, dass jedes Menü gleich aufgebaut sein soll, um einfachere Verwaltung

¹⁶Vgl. Kapitel 6.4.1, Codeausschnitt „Konvertierung Touchscreen zu Screen Koordinaten“

¹⁷Links zu den Bibliotheken in Kapitel 8.3

¹⁸Vgl. Kapitel 6.4.3, Codeausschnitt „Erkennung Tastatur Knopf“

¹⁹Vgl. Abbildung 18

²⁰Kapitel 8.3, Link 1

²¹Konzept der Methode in Abbildung 9, Quellcode in Kapitel 6.4.2, Codeausschnitt „Empfangen von Nachrichten“

²²Konzept der Methode in Abbildung 8, Quellcode in Kapitel 6.4.2, Codeausschnitt „Senden von Nachrichten“

²³Vgl. Kapitel 6.4.2, Codeausschnitt „Konvertierung String zu Char-Array“

²⁴Vgl. Abbildung 10, Quellcode in Kapitel 6.4.2, Codeausschnitt „Überprüfung auf verfügbare Geräte“

²⁵Vgl. Abbildung 6

des Codes zu gewährleisten²⁶. Damit es immer möglich ist, Nachrichten zu empfangen, musste in jedem Menü der Klasse regelmäßig auf eingehende Nachrichten überprüft werden und, sollte eine Nachricht empfangen werden, diese Nachricht, entsprechend des Ziels 6 aus Kapitel 2.2, in einen Nachrichtenspeicher geschrieben werden. Da der Arduino nicht Multithreadfähig ist²⁷, konnte der Prozess nicht parallel laufen, weswegen die Funktion in jedem einzelnen Menü eingebaut werden musste.

Es wurde ein Hauptmenü entwickelt, aus dem heraus ausgewählt werden kann, ob eine Nachricht geschrieben, empfangene Nachrichten im Zwischenspeicher gelesen, das Gerät konfiguriert oder auf Verfügbarkeit anderer Geräte getestet werden soll²⁸. Wird „Schreiben“ ausgewählt, wird eine Methode aufgerufen, die „VKeys“ Klasse initialisiert, wodurch es ermöglicht wird, eine Nachricht zu verfassen. Unter „Optionen“ wurden alle Möglichkeiten der Konfiguration, wie zum Beispiel Anpassung der Farben des Interface durch Instanziierung der „ColorChooser“-Klasse, aber auch die Konfiguration der Distanz durch Einstellen des „Radio“-Objekts. Des Weiteren wurde ein Menüpunkt eingefügt, mit welchem manuell auf verfügbare Geräte in der Nähe geprüft werden kann, welcher wiederum bei Auswahl die entsprechende Methode des „Radio“-Objekts aufruft²⁹.

Der Zwischenspeicher wurde über einen privaten String-Array realisiert, in den jede empfangene Nachricht geschrieben wird. Hier wurde sich entschieden, dem Speicher nur eine Größe von 3 Nachrichten zu geben, um den Speicherplatz des Arduinos zu schonen. Wird eine Nachricht empfangen und der Array ist bereits gefüllt, so wird die älteste Nachricht gelöscht, um Platz für die neue Nachricht zu machen³⁰. Dieser Zwischenspeicher kann im Hauptmenü unter „Lesen“ ausgelesen werden und es besteht die Möglichkeit, alte Nachrichten manuell zu löschen.

Nachdem die „Messenger“-Klasse entwickelt war und alle weiteren Klassen in diese implementiert wurden, wurde das gesamte Projekt noch abschließend getestet und jeglicher Fehler behoben, der bei der Implementierung aufgetreten war.

5 Projektergebnis

5.1 Qualitätskontrolle

Ein wichtiges Qualitätsmerkmal des Projekts ist die korrekte und zuverlässige Übertragung eingetragener Nachrichten. Um dies zu testen, wurden mit jeder der konfigurierbaren Einstellungen der Signalstärke der RF24-API (Min, Low, High und Max) 5 mal die gleiche Nachricht in verschiedenen Distanzen gesendet und gezählt, wie viele der Nachrichten korrekt übertragen wurden³¹. Hier muss allerdings erwähnt werden, dass die Übertragung in einem Gebäude mit groß ausgebautem WLAN Netz getestet wurde, dementsprechend also sowohl Wände, als auch WLAN-Interferenzen das Testergebnis beeinträchtigt haben könnten. Des Weiteren war die Stichprobe relativ klein.

Es ist aber zu erkennen, dass jede der Signalstärkeneinstellungen in mindestens einer der getesteten Distanzen jede Nachricht korrekt übertragen hat, die Qualität der Übertragung also gut ist.

Leider werden, da die Transceiver innerhalb des 2.4GHz Bands arbeiten, auch teilweise Interferenzen als Nachrichten erkannt, was durch die Umsetzung nicht abgefangen wurde.

Des Weiteren sind die Arduinos aktuell noch auf ein Netzteil angewiesen, da ansonsten die Übertragung fehlschlägt, da die 5V Pins des Arduino Mega die Transceiver nicht korrekt zu versorgen scheinen, was durch die Umsetzenden nicht behoben werden konnte. Auch eine angeschlossene Batterie hat leider keine Abhilfe geschaffen. Sind die Netzteile nicht angeschlossen, ist die Übertragung gar nicht möglich oder der Transceiver gibt unregelmäßig Signale an den Arduino weiter, welche wiederum als unleserliche Zeichenketten interpretiert werden³². Dieses Verhalten wird auch durch einen Wackel-

²⁶Vgl. Abbildung 7, Beispielcode eines Menüs in Kapitel 6.4.1, Codeausschnitt „Beispiel Menüführung: Optionenmenü“

²⁷Vgl. Kapitel 3.2.4, Punkt 4

²⁸Vgl. Abbildung 15

²⁹Vgl. Abbildung 10, Quellcode in Kapitel 6.4.2, Codeausschnitt „Überprüfung auf verfügbare Geräte“

³⁰Vgl. Kapitel 6.4.1, Codeausschnitt „Nachrichten zwischenspeichern“

³¹Vgl. Abbildung 13

³²Hier ist keine Abbildung vorhanden, da kein Bild des Displays mit einer solchen Nachricht gemacht wurde.

kontakt des Kabels am 5V Pinout ausgelöst, was wiederum durch Löten der Komponenten behoben werden könnte, allerdings nicht umgesetzt ist.

5.2 Soll-Ist-Vergleich

Im Folgenden wird überprüft, ob die in Kapitel 2.2 erreicht wurden.

Ziel 1 wird durch die Klasse „VKeys“ abgebildet und in der Klasse „Messenger“ implementiert³³. Es ist auch möglich, Sonderzeichen zu schreiben³⁴, das Ziel ist also erfüllt.

Die Ziele 2, 3 und 4 werden durch die Klasse „Radio“ abgebildet und ebenfalls durch die Klasse „Messenger“ implementiert. Die Qualität der Übertragung wurde getestet und das Ergebnis ist zufriedenstellend³⁵. Dieses Ziel ist somit ebenfalls erreicht.

Ziel 5 wird durch die Klasse „Messenger“ abgebildet und implementiert. Das graphische User-Interface hat außerdem eine Menüführung³⁶. Entsprechend kann das Ziel als erreicht bewertet werden.

Ziel 6 wird ebenfalls durch die Klasse „Messenger“ abgebildet und implementiert. Es existiert ein Zwischenspeicher für insgesamt 3 Nachrichten, aus dem die älteste Nachricht entfernt wird, sollte eine weitere Nachricht empfangen werden. Dieses Ziel ist somit abgebildet

Ziel 7 wird auch durch die Klasse „Messenger“ abgebildet und implementiert. Es wurde im Hauptmenü ein eigener Menüpunkt zum Lesen der Nachrichten implementiert, der wiederum in ein eigenes Menü führt³⁷, das Ziel ist also erfüllt.

Auch die optionalen Ziele wurden alle erfüllt. Es ist möglich, das User Interface farblich anzupassen³⁸, es ist möglich, manuell zu prüfen, ob empfangsbereite Geräte in der Nähe sind und es kann über die Optionen die Sendedistanz eingestellt werden.

Zusammenfassend kann also gesagt werden, dass jedes Ziel erfüllt wurde, die Geräte allerdings nicht mobil sind, da noch keine Lösung für das benötigte Netzteil gefunden wurde, wodurch eine der Grundideen des Projekts leider nicht realisiert wurde.

5.3 Ausblick

Es wurde noch keine Halterung / kein Case für das Projekt gebaut, wodurch es aktuell noch nicht robust ist. Sollte das Projekt fortgesetzt werden, wäre dies der nächste Schritt.

Eine interessante Erweiterung für das Projekt wäre außerdem die Entwicklung eines eigenen Formats der Nachrichten oder einem eigenen Protokoll der Übertragung, um sicherzustellen, dass nur Nachrichten, die tatsächlich von Geräten mit der entwickelten Technologie, verarbeitet werden und somit Interferenzen ignoriert werden. So könnten außerdem, zu den Nachrichten gehörende, Metadaten (Uhrzeit, Geräte-ID etc.) übertragen werden, wodurch die Kommunikation deutlich präziser wird und auch nur, sollten mehrere Geräte gleichen Typs in der Umgebung verfügbar sein, ein bestimmtes Gerät angesprochen werden. Ein solches Protokoll würde auch das Senden von Kommandosequenzen vereinfachen.

³³Vgl. Abbildung 18

³⁴Vgl. Abbildung 19

³⁵Vgl. Abbildung 13

³⁶Vgl. Abbildung 15

³⁷Vgl. Abbildung 20

³⁸Vgl. Abbildung 17 und Abbildung 16

6 Anlagen

6.1 Auflistung der Komponenten

Tabelle 1: Benötigte Hardware

Hardware	Aufgabe	Kosten
Arduino Mega 2560	- Zentrale Schnittstelle aller Komponenten - Programmierbarkeit, Verwaltung der Logik	2 x 21,99€
Elegoo Uno TFT Touchscreen 2,8"	<ul style="list-style-type: none"> Anzeige von Nachrichten Eingabe von Nachrichten User Interface 	2 x 19,99€
nRF24L01+ Wireless Transceiver Modul	<ul style="list-style-type: none"> Nachrichten Übertragen und Empfangen Überprüfung Verfügbarkeit anderer Geräte 	2 x 5,-€

Gesamtkosten: 93.96€.

Links zur genutzten Hardware sind in Kapitel 8.2 aufgeführt.

Tabelle 2: Benötigte Software

Bibliothek	Aufgabe	Quelle
Elegoo_GFX.h	Kern Grafikbibliothek des Elegoo Uno TFT Touchscreens. Ermöglicht das Drucken von Zeichen / Formen auf TFT Display.	Mitgeliefert auf CD bei TFT Touchscreen
Elegoo_TFTLCD.h	Hardware-Bibliothek des Elegoo Uno TFT Touchscreens. Verantwortlich für die Kommunikation des Programms mit der Hardware.	Mitgeliefert auf CD bei TFT Touchscreen
TouchScreen.h	Bibliothek des Touchscreens des Elegoo Uno TFT Touchscreens. Erlaubt das erkennen von Berührungen des Touchscreens und die Lokalisierung der Berührung.	Mitgeliefert auf CD bei TFT Touchscreen
SPI.h	Erlaubt die Kommunikation des Programms mit dem SPI Bus des Arduino Board	In Arduino IDE inkludiert
nRF24L01.h	Hardware Bibliothek der nRF24L01+ Transceiver. Erlaubt Kommunikation des Moduls mit dem Arduino Board	Github
RF24.h	Programmierbare Schnittstelle der nRF24L01+ Transceiver.	Github
Arduino.h	Liefert Kernfunktionen der Arduino Boards.	In Arduino IDE inkludiert

Links zur genutzten Software sind in Kapitel 8.3 aufgeführt.

6.2 Diagramme

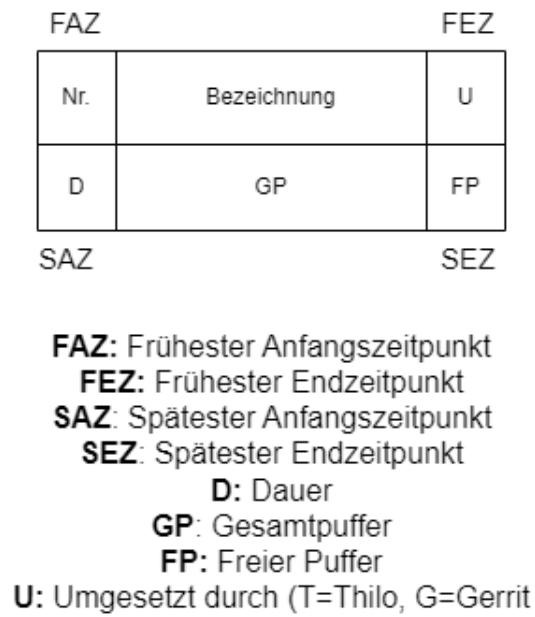


Abbildung 1: Legende der Netzplan Knoten

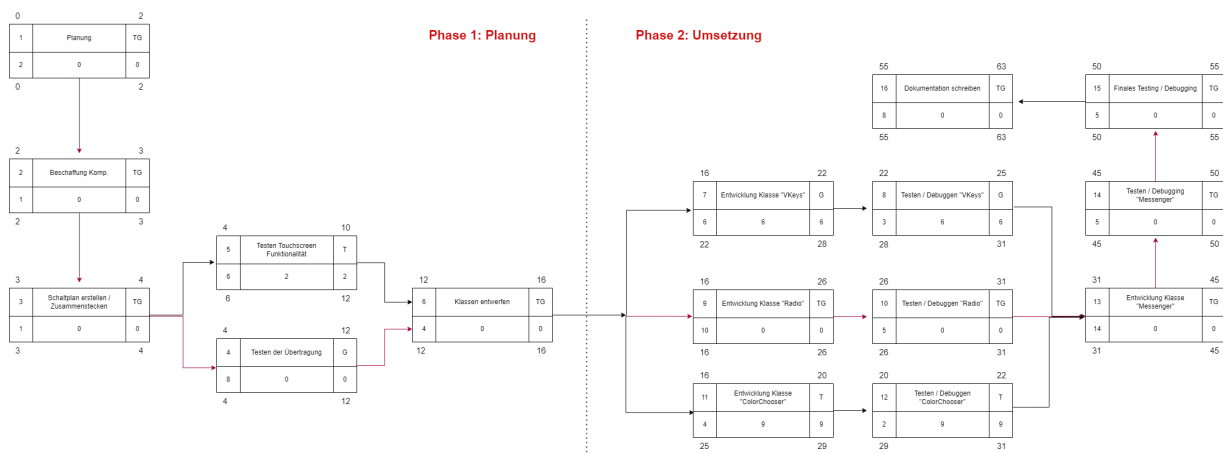


Abbildung 2: Netzplan des Projekts

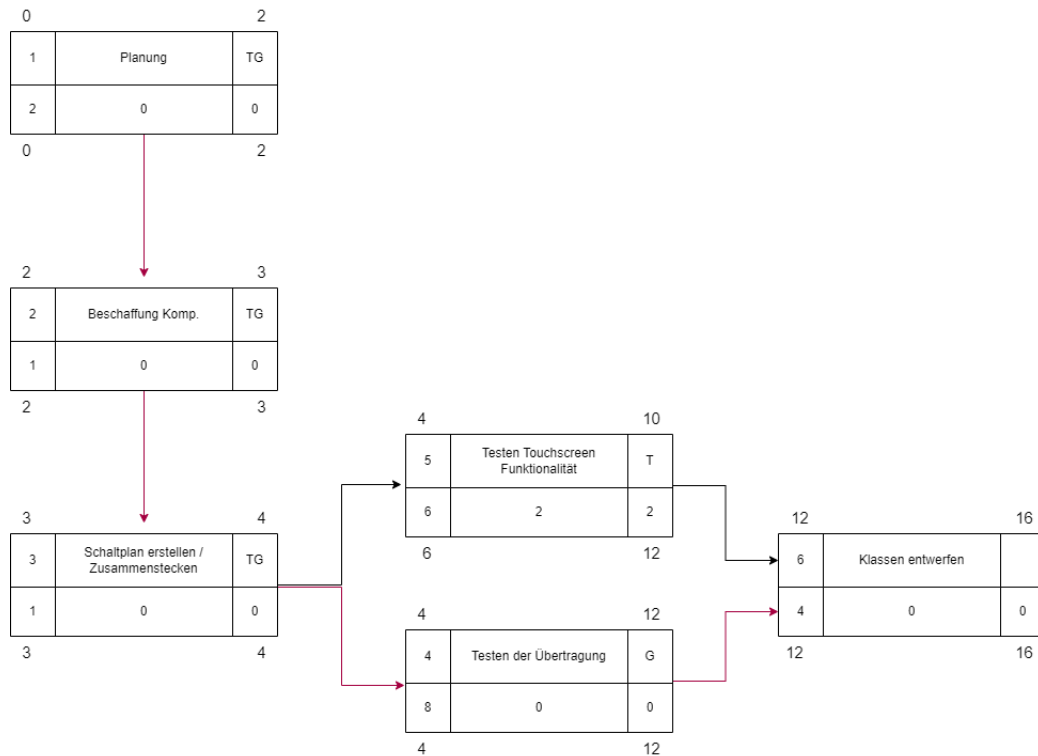


Abbildung 3: Netzplan der Projektplanung

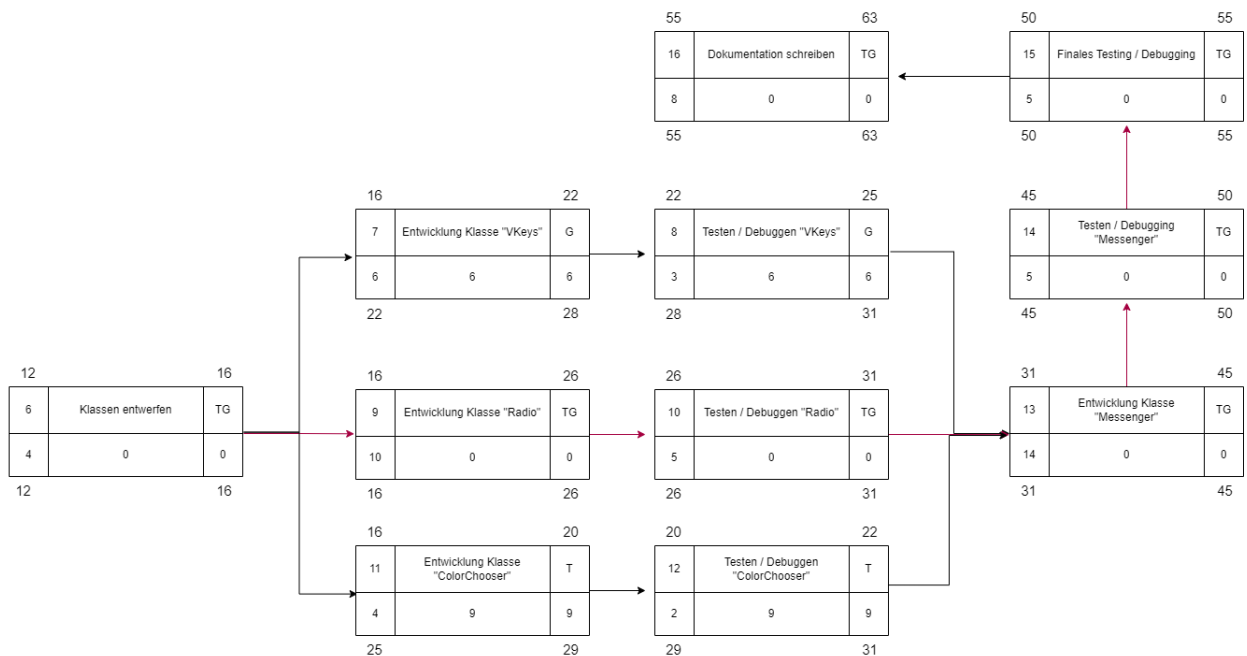


Abbildung 4: Netzplan der Projektumsetzung

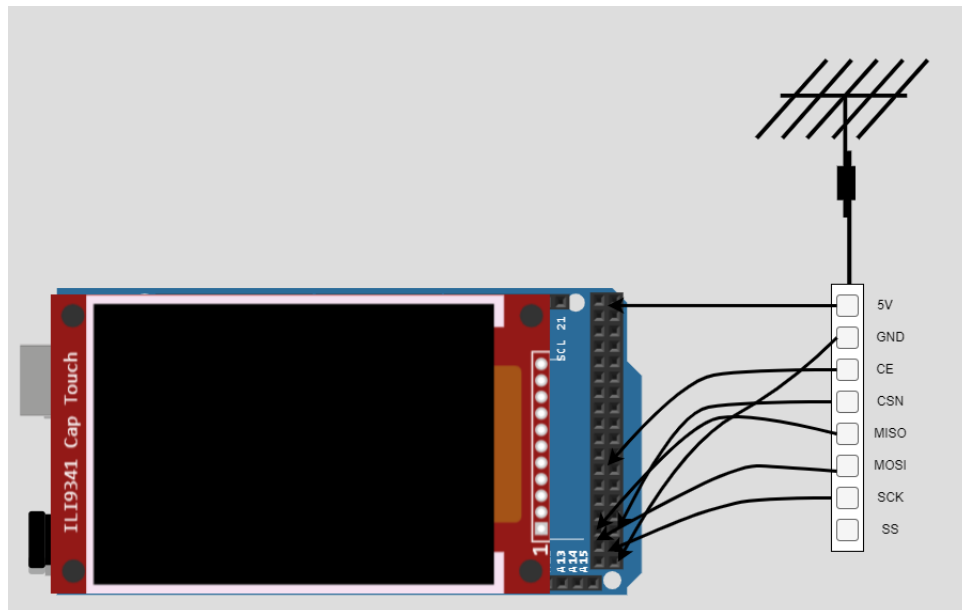


Abbildung 5: Schaltplan des Projekts

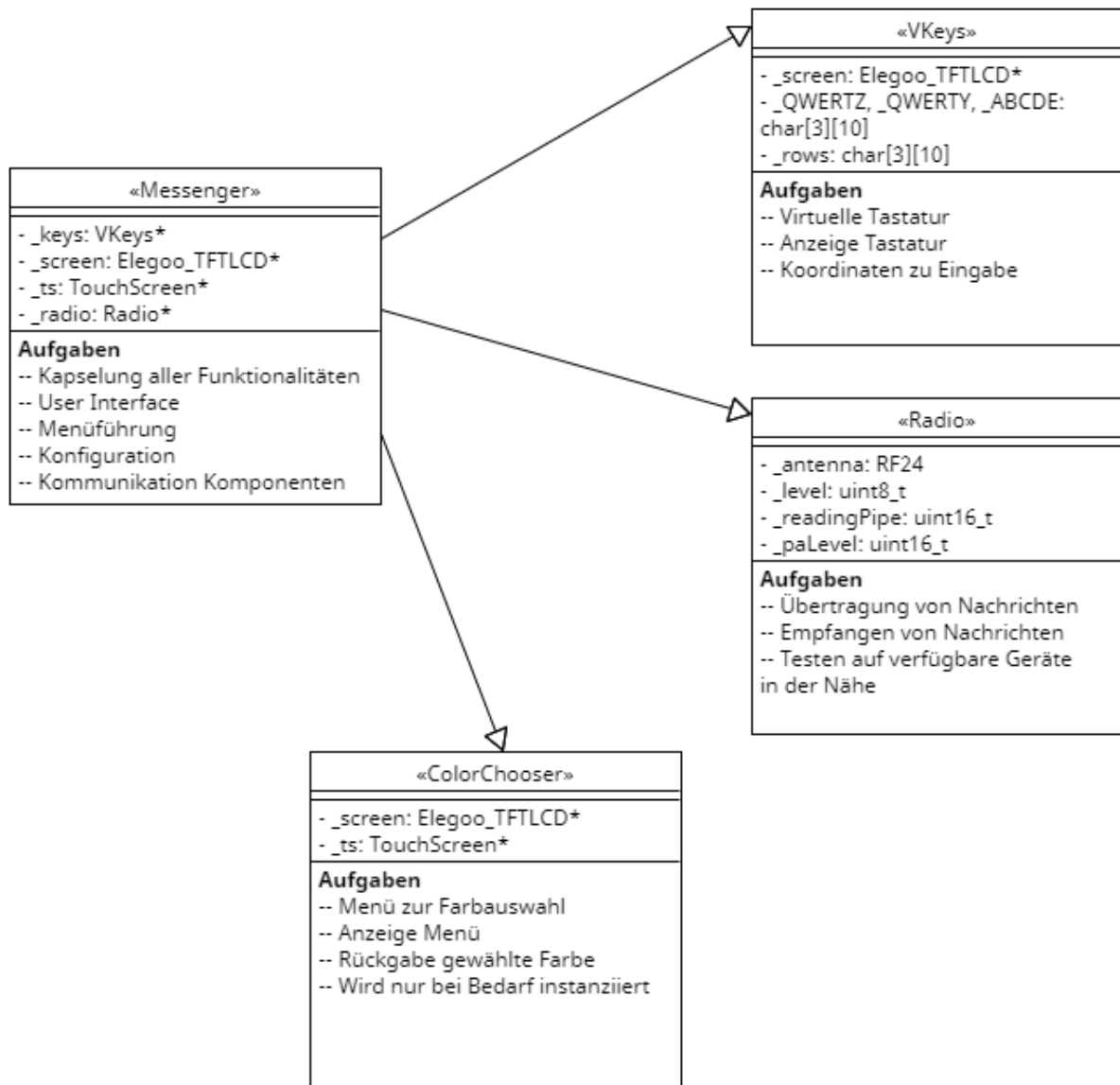


Abbildung 6: Vereinfachtes Klassendiagramm des Arduino Programms

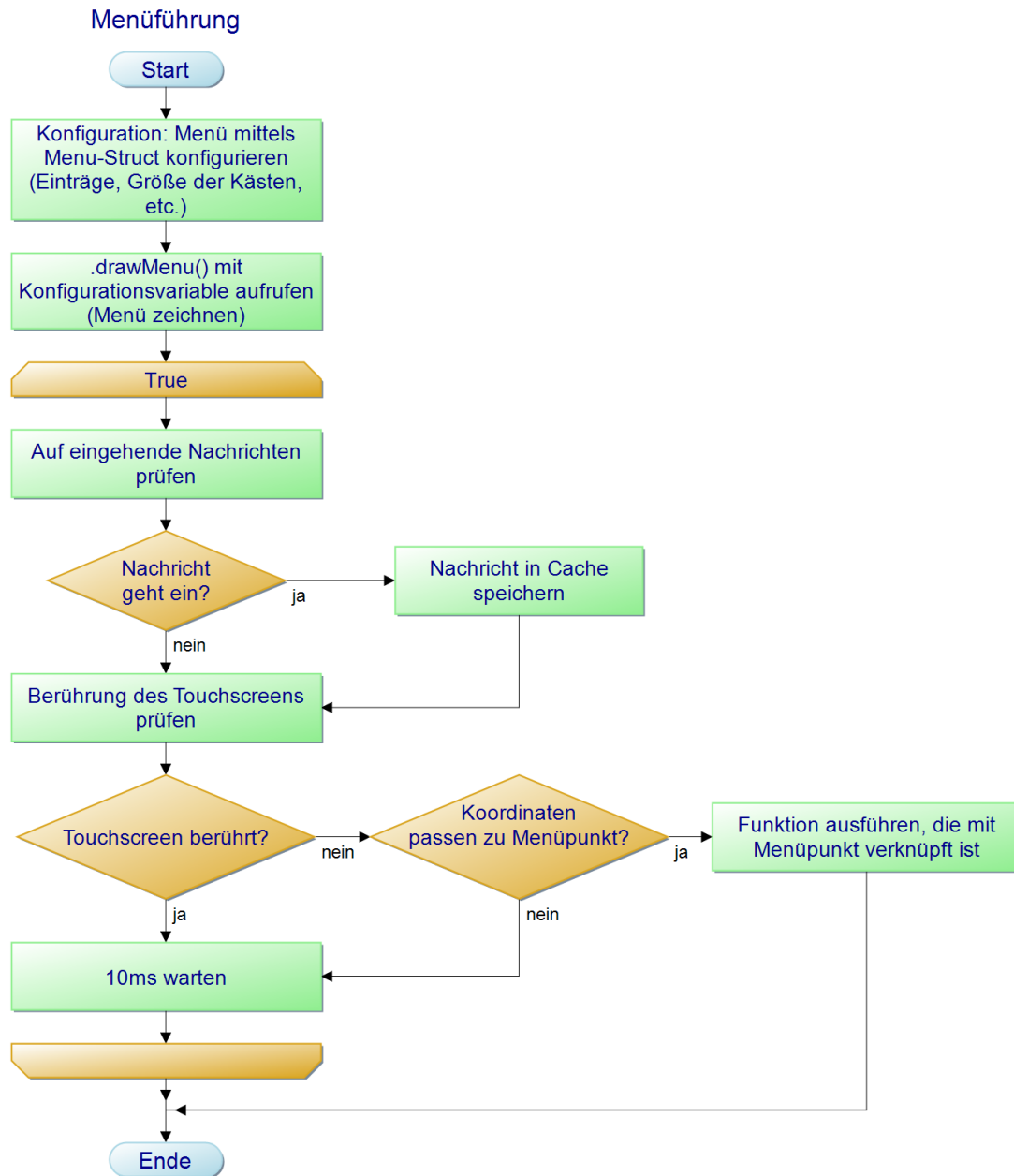


Abbildung 7: Aufbau der Menümethoden

Nachrichten empfangen

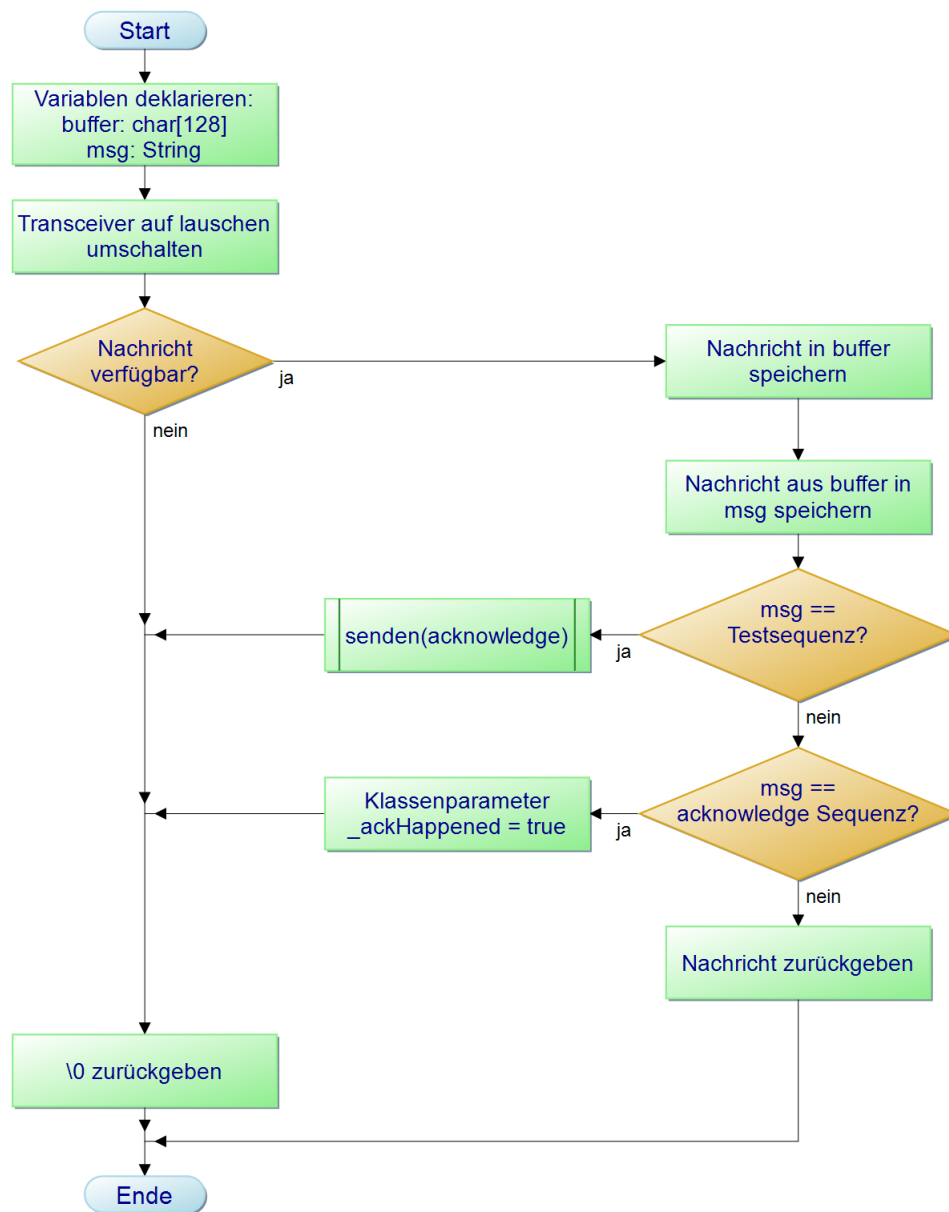


Abbildung 8: Ablauf des Nachrichtenempfangs

Nachricht senden (String msg)

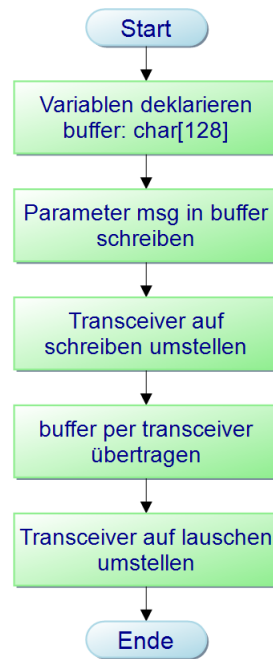


Abbildung 9: Ablauf des Sendens von Nachrichten

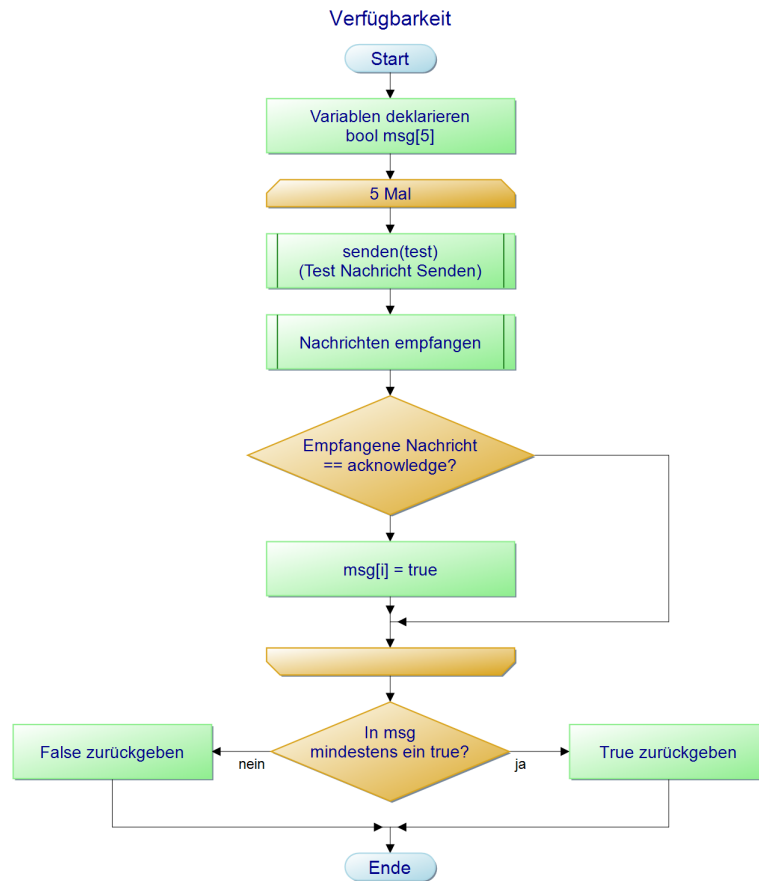


Abbildung 10: Ablauf des Testens auf verfügbare Geräte

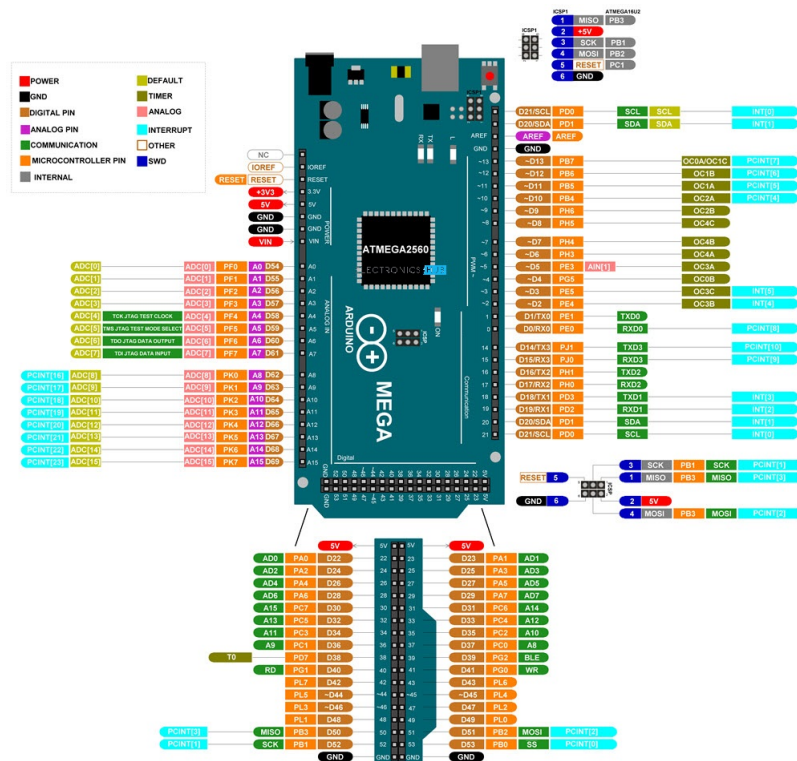


Abbildung 11: Pinout Beschreibung des Arduino Mega 2560 R3, Quelle: Internetquelle 4

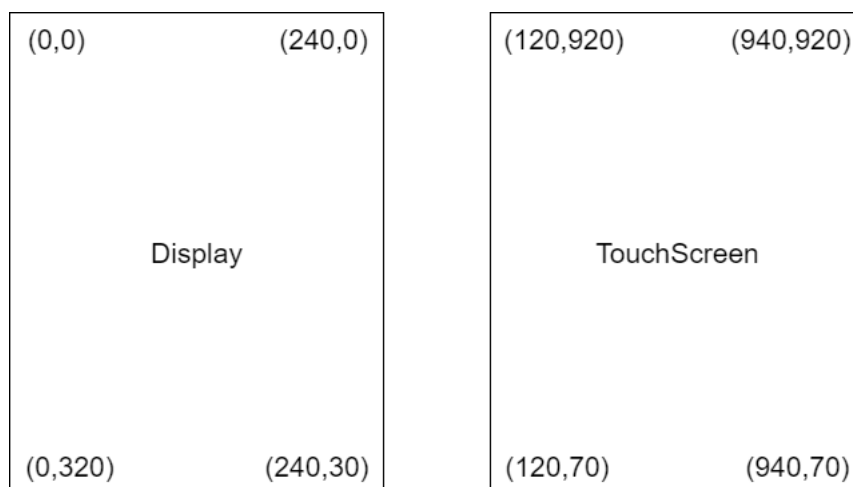


Abbildung 12: Darstellung der Koordinaten von Touchscreen und Display (x,y)

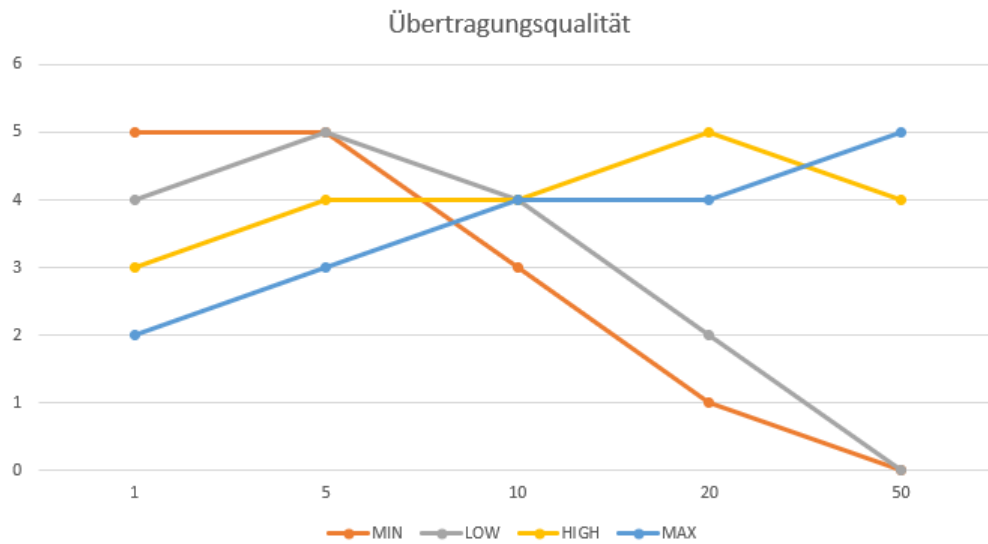


Abbildung 13: Ergebnis Nachrichtenübertragung. X: Distanz in Metern; Y: Korrekt übertragene Nachrichten von 5 gesendeten.

6.3 Bilder

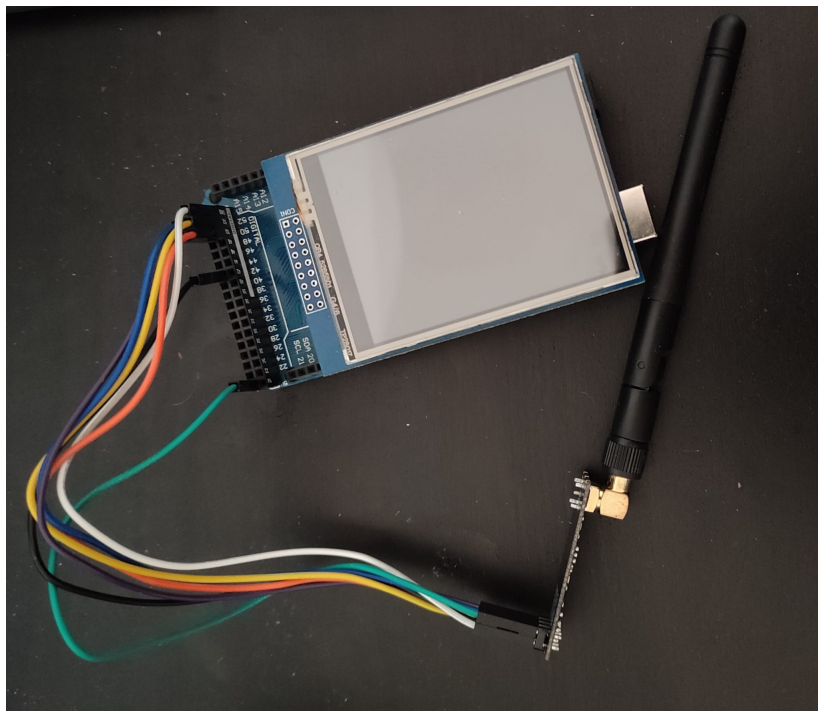


Abbildung 14: Bild des aufgebauten Projekts



Abbildung 15: Standard-Hauptmenü des Projekts

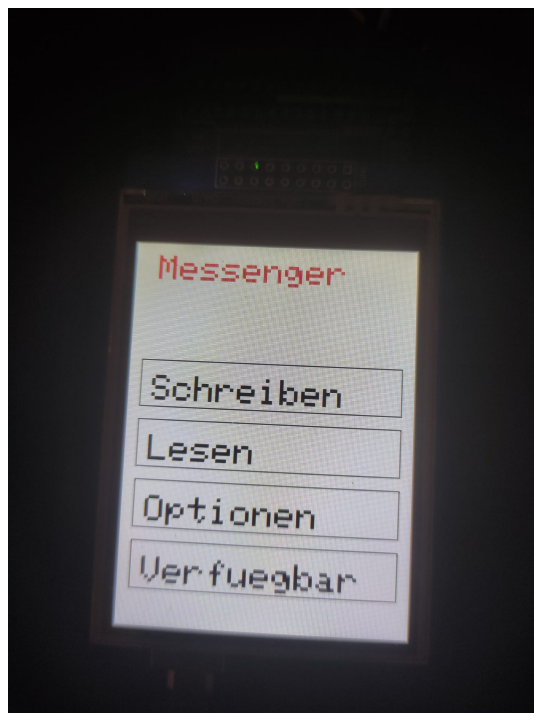


Abbildung 16: Hauptmenü des Projekts mit Lightmode



Abbildung 17: Virtuelle Tastatur des Projekts

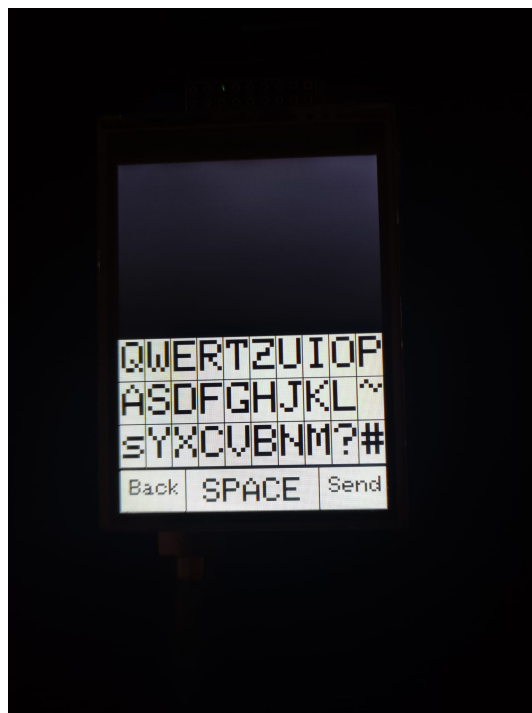


Abbildung 18: Virtuelle Tastatur des Projekts



Abbildung 19: Virtuelle Tastatur des Projekts

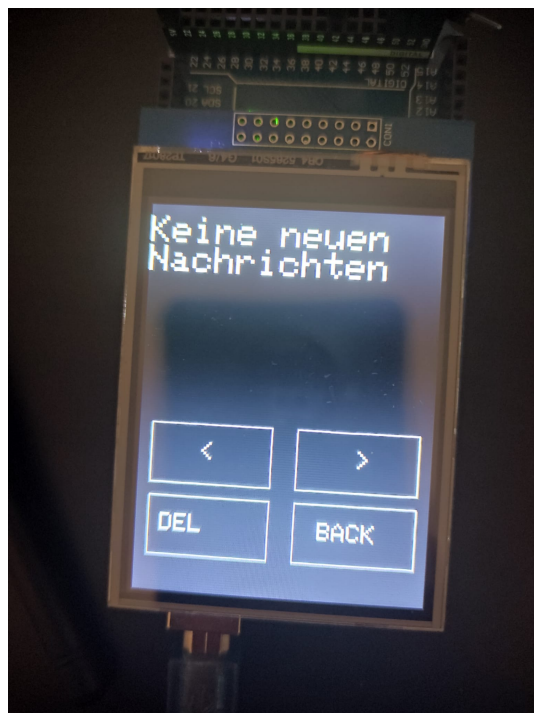


Abbildung 20: Virtuelle Tastatur des Projekts

6.4 Quellcode Auszüge

Der gesamte Quellcode ist in einer Zip-Datei beigelegt.

Im Folgenden sind in der Dokumentation benannte Funktionen und Methoden angelegt.

6.4.1 Klasse: Messenger

Konvertierung Touchscreen zu Screen Koordinaten

```
/**
 * @brief
 * Parses Coordinates from touchpoint to screen coordinates
 *
 * @param p TSPoint, Point the user touched on display
 * @return ScreenParse Struct containing parsed screen coordinates
 */
ScreenParse Messenger::parseCoords(TSPoint p) {
    ScreenParse parse;

    // Touch X: [120,940], Screen X: [0,240]
    // Parse X
    parse.x = (int) ((p.x - 120) / 3.5);

    // Touch Y: [70,920], Screen Y: [0,320]
    // Parse Y
    parse.y = (int) ((p.y - 70) / 2.66);
    return parse;
}
```

Beispiel Menüführung: Optionenmenü

```
/**
 * @brief
 * Options menu
 *
 */
void Messenger::optsMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 40;
    menu.menuOffset = 20;
    menu.header = "Optionen";
    menu.entries[0] = String("Farben");
    menu.entries[1] = String("Tastatur");
    menu.entries[2] = String("Distanz");
    menu.entries[3] = String("Zurueck");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);
        int selection = -1;

        // Get touchpoint
```



```

digitalWrite(13, HIGH);
TSPoint p = _ts->getPoint();
digitalWrite(13, LOW);

// If touch was recognized
if (p.z > _minTouch) {
    // Get selected menu poin
    selection = getSelection(menu.menuStart, menu.menuThickness,
        menu.menuOffset, 4, parseCoords(p));

    Serial.println("Selection: " + String(selection));

    switch (selection) {
        case -1: break;
        case 1: colorMenu(); return;
        case 2: keysMenu(); return;
        case 3: distanceMenu(); return;
        case 4: return;
        default: break;
    }
}
delay(10);
}

Serial.println("Returning from optsMenu");
}

```

Nachrichten zwischenspeichern

```

/**
 * @brief
 * Caches incoming messages in message cache
 *
 * @param msg String to be cached
 */
void Messenger::cacheMessage(String msg) {
    // Check, how many messages are in cache
    int counter = 0;
    for (int i = 0; i < 3; i++) {
        if (_messages[i] != "\0") {
            counter++;
            continue;
        } else {
            break;
        }
    }

    // If message already is in cache, return
    for (int i = 0; i < counter; i++) {
        if (msg == _messages[i]) return;
    }

    // If there are less than 3 messages, just cache the newest message on last
    position
    if (counter < 3) {
        _messages[counter] = msg;
        return;
    } else {

```

```

    // If there are 3 messages, clear the last message
    _messages[0] = "\0";

    // Push every other message one to the back
    for (int i = 1; i < 3; i++) {
        _messages[i - 1] = _messages[i];
    }

    // Put new Message as first in the array
    _messages[2] = msg;
    return;
}
}

```

6.4.2 Klasse: Radio

Empfangen von Nachrichten

```

/**
 * @brief
 * Receives message from radio frequency
 *
 * @return String received message or \0, if nothing was gotten
 */
String Radio::receiveMessage(void) {
    char buffer[128] = "";
    String msg = "";

    // Switch from sending to listening
    if (!_listening) switchState();

    // If no message is available, return universal break character
    if (!_antenna.available()) {
        //Serial.println("Receiving: No Antenna available");
        return "\0";
    }
    Serial.println("Receiving: Antenna available");

    // Read available payload
    _antenna.read(&buffer, sizeof(buffer));

    for (int i = 0; i < 128; i++) {
        msg += String(buffer[i]);
        if (buffer[i] == '\0') break;
    }

    Serial.println("Receiving: Received char Array: " + msg);

    // If the test string was received, acknowledge it and return \0
    if (msg == String(_test)) {
        Serial.println("Receiving: Test string, now acknowledging.");
        acknowledge();
        return "\0";
    }

    // If acknowledge string was received, change marker in class fields to true so it
    // can be checked by other methods and return \0
    if (msg == String(_acknowledge)) {
        Serial.println("Receiving: Acknowledge String");
    }
}

```

```

        _ackHappened = true;
        return "\0";
    }

    return msg;
}

```

Senden von Nachrichten

```

/**
 * @brief
 * Sends message given in param
 *
 * @param msg Message to be sent
 * @return true If sending was successful
 * @return false otherwise
 */
bool Radio::sendMessage(String msg) {
    String sendTest = "";
    bool tx_ok, tx_fail, rx_ready, test = true;
    char buffer[128];
    msg.toCharArray(buffer, msg.length() + 1);

    Serial.println("Sending message: " + msg);
    Serial.println("Listening State: " + String((_listening ? "listening" :
        "writing")));

    if (_listening) switchState();

    Serial.println("Listening State: " + String((_listening ? "listening" :
        "writing")));

    // convertStringToCharArray(msg, buffer);
    for (int i = 0; i < msg.length(); i++) {
        sendTest += String(buffer[i]);
    }
    Serial.println("Sending: Char Arr after conversion: " + sendTest);

    test = _antenna.write(&buffer, sizeof(buffer));

    if (!test) {
        Serial.println("Sending unseccessful");
        _antenna.whatHappened(tx_ok, tx_fail, rx_ready);
        Serial.println("OK: " + String((tx_ok ? "Yes" : "No")) + ", FAIL: " +
            String((tx_fail ? "Yes" : "No")) + ", READY: " + String((rx_ready ? "Yes"
                : "No")));
        switchState();
        return false;
    }

    Serial.println("Returning from sending");
    switchState();
    return true;
}

```

Überprüfung auf verfügbare Geräte

```

/**
 * @brief
 * Checks, whether there are devices available nearby
 *
 * @return true If there are devices available nearby
 * @return false otherwise
 */
bool Radio::checkNearbyDevices(void) {
    _ackHappened = false;
    bool msg[5] = {false, false, false, false, false};

    for (int i = 0; i < 5; i++) {
        sendMessage(_test);
        receiveMessage();

        msg[i] = wasAcknowledged();
        Serial.println("Msg " + String(i) + ": " + String((msg[i] ? "True" :
            "False")));
        delay(1000);
    }

    // If payload was the acknowledge String, that means there are devices nearby.
    for (int i = 0; i < 5; i++) {
        if (msg[i]) return true;
    }

    Serial.println("_ack: " + String((_ackHappened ? "True" : "False")));
    _ackHappened = false;
    return false;
}

```

Konvertierung String zu Char-Array

```

/**
 * @brief
 * Converts a given String s to a char array and saves it to given pointer a
 *
 * @param s String to be converted to char array
 * @param a Pointer to char array
 */
void Radio::convertStringToCharArray(String s, char a[128]) {
    int counter = 0;

    // Iterate through entire String and add character to char array
    for (int i = 0; i < s.length(); i++) {
        a[i] = s.charAt(i);
        counter++;
    }

    // Fill up char array with empty values
    for (int i = counter; i < 128; i++) {
        a[i] = '\0';
    }

    Serial.println("Char Arr after conversion: " + String(a));
}

```

Wechsel zwischen senden und lauschen

```

/**
 * @brief
 * Switches from listening to sending and the other way around
 *
 */
void Radio::switchState() {
    byte _address[6] = "00001";
    // Invert listening
    _listening = !_listening;

    // If listening: Prepare antenna for listening. Otherwise: Prepare antenna for
    // sending
    if (_listening) {
        _antenna.openReadingPipe(_readingPipe, _address);
        _antenna.startListening();
    } else {
        _antenna.stopListening();
        _antenna.openWritingPipe(_address);
    }

    _antenna.setPALevel(_level);
}

```

6.4.3 Klasse: VKeys

Erkennung Tastatur Knopf

```

/**
 * @brief
 * Returns selected char by calculating, which column and row was pressed from passed
 * x and y coords
 *
 * @param x X coordinate from touchscreen selection
 * @param y Y coordinate from touchscreen selection
 * @return String Selected character parsed to a string, BACK if back button was
 *         selected and DONE, if done button was selected
 */
String VKeys::getCharFromCoords(int16_t x, int16_t y) {
    int row, column;

    // Complete Y: [70,920], range 850, keyboard [70,475], keyboard range of 405, 4
    // rows so 405/4 = 101.25
    if (y > 475) return String('\0');
    row = (int) ((y - 70) / 101.25);

    // row 0 is spacebar, therefore return escaped space immediately, as nothing else
    // is there.
    if (row == 0) {
        int tempX = (int)((x - 120) / 3.5);

        if (tempX < (_screen->width() / 4)) {
            return String("BACK");
        } else if (tempX > 3*(_screen->width() / 4)) {
            return String("DONE");
        }
        return String(" ");
    }
}

```

```

}

if (row > 3) return String('2');

// X Range ca. 120 - 920 for 10 columns, therefore 800 / 10 = 80
column = (int) ((x - 120) / 80);

if (column < 11 && column >= 0 && row < 4 && row > 0) return String(_special ?
    _specialChars[(3-row)][column] : _rows[(3-row)][column]);

return String('-');
}

```

6.4.4 Klasse: ColorChooser

Erkennen der ausgewählten Farbe

```

/**
 * @brief
 * Analyses coordinates passed into function and returns selected color
 *
 * @param coords Coordinates of touchpoint, parsed to screen coordinates
 * @return uint16_t chosen Color or 0, if cancelled or -2, if selection was not valid
 */
uint16_t ColorChooser::getSelection(Coords coords) {
    int column, row;

    Serial.println("CC: X: " + String(coords.x) + ", Y: " + String(coords.y));

    // Check, if touchpoint is at height of cancel button
    if (320 - coords.y > 250) return 0;

    // If coordinates are out of valid ranges, return failure marker
    if (coords.x < 35 || coords.x > 215) return -2;
    if (coords.x < 135 && coords.x > 100) return -2;

    // If x is less than 100, it has to be first column (index 0), otherwise second
    // column (index 1)
    column = coords.x < 100 ? 0 : 1;

    // Invert coordinates (320-y, because screen is 320 high) and divide by 60,
    // because that's the
    // height of the boxes, to get the row
    row = (int)((320 - coords.y) / 60);

    // Variable basically only exists, so that we don't try to reach indices the array
    // does not have
    // This works, because the first column has indices 0,2,4,6 and second has 1,3,5,7
    int safeHandler = column + (row*2);

    // If selection > 7, set selection to failure, otherwise set selection to selected
    // color
    uint16_t selection = (safeHandler > 7 ? -2 : _colors[safeHandler]);

    return selection;
}

```

7 Glossar

7.1 Technische Begriffe

Multithreading

Unter Multithreading versteht man in der Informatik den Prozess, ein Programm in mehrere Teilstränge aufzuteilen, die parallel ausgeführt werden.³⁹

SPI-Bus

SPI steht für Serial Peripheral Interface und ist ein, aus drei Leitungen bestehendes, Bussystem nach dem Master-Slave Prinzip, mit dem die Kommunikation verschiedener Komponenten gesteuert werden kann.⁴⁰

Bottom-Up

Der Bottom-Up Ansatz beschreibt in der Software-Entwicklung die Vorgehensweise, zunächst Teilfunktionen oder einzelne Klassen zu entwickeln und diese, nach ihrer Fertigstellung, zu einem System zusammenzuführen. Im Gegensatz hierzu wird bei der Top-Down Methode zunächst das System definiert, ohne im Detail auf einzelne Funktionen einzugehen.⁴¹

7.2 Arduino Begriffe

Shield

Ein Shield ist eine Schaltplatte, die die Funktionalität eines Arduinos erweitern kann und meistens so konzipiert ist, dass sie direkt in den Arduino gesteckt werden kann und diesen somit zum Teil oder vollständig abdeckt.⁴²

³⁹Vgl. Internetquelle 1

⁴⁰Vgl. Internetquelle 3

⁴¹Vgl. Internetquelle 5

⁴²Vgl. Internetquelle 2

8 Quellenverzeichnis

8.1 Internetquellen

1. Storage Insider: Was ist Multithreading - Online unter <https://www.storage-insider.de/was-ist-multithreading-a-1017586/> [07.01.2023]
2. Computer Hope: Arduino Shield - Online unter [07.01.2023]
3. Mikrocontroller.net: Serial Peripheral Interface - Online unter https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface [15.01.2023]
4. Electronics Hub: Arduino Mega Pinout - Online unter <https://www.electronicshub.org/wp-content/uploads/2021/01/Arduino-Mega-Pinout.jpg> [15.01.2023]
5. Geeks for Geeks: Difference between Bottom-Up Model and Top-Down Model - Online unter <https://www.geeksforgeeks.org/difference-between-bottom-up-model-and-top-down-model/> [18.01.2023]

8.2 Links genutzter Hardware

1. nRF24L01+ Transceiver:
2. Elegoo Arduino Mega R3:
3. Elegoo TFT Touchscreen:

8.3 Links genutzter Software

1. RF24: Online unter <https://github.com/nRF24/RF24>