Hochschule Anhalt
Anhalt University of Applied Sciences

Projektdokumentation

LED Logikspiel

Eingereicht von:	Marcus Gagelmann
Matrikelnummer:	504 11 61
Studienfach:	Interaktive Mediensysteme
Betreuender Hochschullehrer:	Prof. Dr. Alexander Carôt
Tag der Einreichung:	22. November 2022

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Halle, 12. November 2022

Marcus Gagelmann

Kurzfassung

Das Logikspiel *Mastermind* war eines der beliebtesten Spiele der 1970er Jahre [1]. Seitdem wurde das Spielkonzept bis heute viele Male in anderen Spielen wiederverwendet und adaptiert. Diese Arbeit dokumentiert den Versuch, das Spielprinzip von *Mastermind* mithilfe eines Arduino Starter Kits in ein Singleplayer-Spiel gegen ein Programm zu übertragen. Das Ziel des Spielers ist es, eine zufällig generierte, vierstellige Zahlenfolge zu erraten. Hierfür wird dem Nutzer nach jedem Rateversuch Feedback zu der Korrektheit der gewählten Ziffern und Zifferpositionen in der Zahlenfolge mithilfe von LEDs vermittelt.

Inhaltsverzeichnis

1 Motivation und Einleitung **1**

2 Konzeption und Implementierung **4**

2.1 Spielkonzept 4

2.1.1 Neues Spiel 4

2.1.2 Laufendes Spiel 4

2.2 Anforderungsanalyse 6

2.2.1 Funktionale Anforderungen 6

2.2.2 Nichtfunktionale Anforderungen 6

2.3 Hardware-Setup 7

2.4 Implementierung 8

2.4.1 Setup-Funktion 8

2.4.2 Loop-Funktion 9

3 Evaluation und Fazit **18**

Abbildungsverzeichnis **I**

Literaturverzeichnis **II**

1 Motivation und Einleitung

Im Jahre 1971 erfand Mordecai Meirowitz das Gesellschaftsspiel *Mastermind*. Das Spiel wurde das erfolgreichste Spiel der 1970er Jahre mit über 55 verkauften Kopien und gewann darüber hinaus 1973 den ersten Game of the Year Award [1].

Ziel des Spiels ist es, eine vom Gegenspieler festgelegte vierstellige Farbsteckerfolge durch sukzessive Vermutungen zu erraten. Nach jedem Rateversuch gibt der Gegenspieler Feedback dazu, wie viele Stecker des aktuellen Versuchs mit der richtigen Farbe an der korrekten Stelle stecken, wie viele mit einer richtigen Farbe an der falschen Stelle stecken und wie viele nicht in der geheimen Steckerfolge vorkommen. Falls es mehrere Stecker der gleichen Farbe gibt, so wird zu maximal so vielen Steckern eines Versuchs positives Feedback gegeben, wie Stecker dieser Farbe im geheimen Code vorkommen. Der Versuch rot-rot-rot-blau würde so zum Beispiel bei dem geheimen Code rot-rot-blau-blau zu drei korrekten und einem falschen Stecker führen.

Das Feedback erfolgt mithilfe kleiner roter und weißer Stecker, wie in Abbildung 1 zu erkennen ist. Der Spieler hat insgesamt 12 solcher Anläufe, um Informationen zu sammeln und die geheime Steckerfolge möglichst schnell herauszufinden.

Das Konzept von *Mastermind* wurde in den folgenden Jahren nach dessen Veröffentlichung in vielen verschiedenen Varianten adaptiert. Neben vielen Neuveröffentlichungen unter neuem Namen, wie beispielsweise *Logiktrainer* und *SuperHirn* [3], gab es auch Varianten für bis zu fünf Spieler und später auch elektronische Handheld-Versionen [2] (siehe Abbildung 2).

Im Rahmen dieses Projekts wird ein Logikspiel mithilfe eines Arduino Starter Kits entwickelt, welches sich am Konzept von *Mastermind* orientiert. Ein einzelner Spieler soll hierbei eine Ziffernfolge erraten. Der Gegenspieler wird durch das Programm repräsentiert, welches der Arduino ausführt. Dieses Programm generiert den geheimen Ziffern-Code und gibt dem Spieler mithilfe von LEDs Feedback zu seinen Rateversuchen. Nach einer festgelegten Anzahl von Versuchen hat der Spieler den Code entweder erraten oder das Spiel verloren und kann mit einem neuen Spiel beginnen.

Abbildung 1: Foto eines vollendeten Mastermind-Spiels
Quelle: [2]

Abbildung 2: Foto eines elektronischen Mastermind-Spiels
Quelle: [2]

2 Konzeption und Implementierung

Im Folgenden wird die Realisierung des zu entwickelnden Arduino-Logikspiels beschrieben. Hierbei wird insbesondere auf das Spielkonzept, die zugrundeliegenden Anforderungen und das dafür benötigte Hardware-Setup eingegangen. Im Anschluss werden Einblicke in die Umsetzung des Programmcodes sowie in aufgetretene Problemen während der Entwicklung gewährt.

2.1 Spielkonzept

Das System des Spiels befindet sich, falls dieses mit Strom versorgt wird, zu jedem Zeitpunkt in einem von zwei verschiedenen Zuständen. Diese Zustände werden im folgenden Abschnitt genauer erklärt.

2.1.1 Neues Spiel

Jede neue Spielrunde des Arduino Logikspiels beginnt mit der Generierung einer zufälligen vierstelligen Ziffernfolge, welche lediglich innerhalb des Arduino-Speichers vorliegt und dem Spieler vorenthalten wird. Das Ziel des Spiels ist es, diesen Code sukzessiv zu erraten.

Wurde der Arduino neu gestartet, das vorherige Spiel abgeschlossen, oder ein neues Spiel mithilfe der Steuertasten initiiert, so befindet sich das System im Zustand „Neues Spiel“. Dieser Zustand kann anhand der weißen Status-LEDs erkannt werden und liegt vor, wenn die linke Status-LED leuchtet und die rechte Status-LED nicht leuchtet. Der Spieler hat nun die Möglichkeit, Ziffern einzugeben und damit seinen ersten Rateversuch des neuen Spiels durchzuführen. Durch das Betätigen einer beliebigen roten Taste des Membrane Switches (siehe Abbildung 3) kann das Spiel zu jeder Zeit zurückgesetzt und ein neues Spiel initiiert werden.

2.1.2 Laufendes Spiel

Sobald der Spieler die erste Ziffer eines neuen Spiels eingibt, wechselt der Spielzustand von „Neues Spiel“ zu „Laufendes Spiel“. Dieser Zustand kann vom Nutzer durch eine dunkle linke Status-LED und eine helle rechte Status-LED erkannt werden.

In diesem Spielzustand muss der Spieler über mehrere Versuchsrunden hinweg Informationen über die geheime Ziffernfolge innerhalb des Programms sammeln und schlussendlich, in weniger als sechs Versuchen, den gesuchten Code über den Membrane Switch eingeben. Sobald der Nutzer insgesamt vier Ziffern an das Programm übergeben hat, wertet dieses die Eingabe aus. Hierfür wird vom Programm für jede der vier Ziffern Feedback in Form einer

Abbildung 3: Foto des verwendeten Membrane Switches
Quelle: eigene Darstellung

leuchtenden LED innerhalb derjenigen LED-Ampel (siehe Abbildung 4) gegeben, die der jeweiligen Zifferposition zugeordnet ist.

Gleiche Ziffern, die sowohl im Versuchs-Code als auch im geheimen Code an der gleichen Stelle stehen, lösen dabei das Leuchten einer grünen LED aus. Ziffern, welche zwar in ihrer Art mit einer Ziffer im gesuchten Code übereinstimmen, aber nicht an der korrekten Position stehen, werden mit einer gelben LED markiert. Schlussendlich lösen Ziffern, die in ihrer Art mit keiner Ziffer aus dem gesuchten Code übereinstimmen, eine rote LED aus.

Nähert sich der Spieler seinem Versuchslimit, so wird dies durch ein Aufblinken der linken Status-LED angezeigt, direkt nachdem ein vollständiger Versuchs-Code eingegeben wurde. Hat der Spieler noch drei Versuche übrig, dann blinkt die linke Status-LED insgesamt dreimal auf. Bei zwei verbleibenden Versuchen blinkt sie zweimal und vor dem letzten Versuch blinkt sie einmal auf.

Schafft es der Spieler nicht innerhalb der vorgegebenen sechs Versuche den korrekten Code zu finden, dann hat er das Spiel verloren. Dies wird durch ein 2,5 Sekunden langes Blinken aller roten LEDs der LED-Ampeln angezeigt. Findet der Spieler den Code wiederum, bevor er alle seine Versuche aufgebraucht hat, so blinken stattdessen alle grünen LEDs der LED-

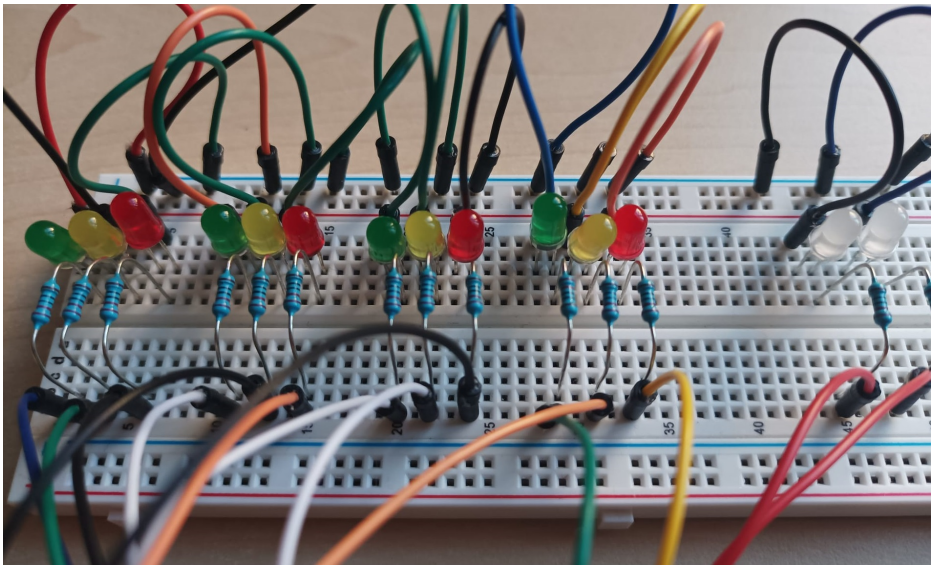

Abbildung 4: Foto der LED-Ampeln und Status-LEDs
Quelle: eigene Darstellung

Ampeln für 2,5 Sekunden. In jedem Fall wechselt nach einem Spielende der Spielstatus zu „Neues Spiel“.

2.2 Anforderungsanalyse

Die im Folgenden aufgezählten Anforderungen an das Projekt lassen sich in funktionale und nichtfunktionale Anforderungen unterteilen.

2.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die gewünschten Funktionalitäten des Spiels. Die funktionalen Anforderungen für das Projekt sind folgende:

- **Membrane Switch** erfasst die Ziffern- und Steuereingaben des Spielers
- ***Arduino*** leistet die Rechenarbeit
- **Software** gibt die Spiellogik vor
- **LEDs** in unterschiedlichen Farben geben Feedback

2.2.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beschreiben die Qualität der Leistungen, welche das System erbringen soll. Die folgenden nichtfunktionalen Anforderungen sind für das Projekt essenziell:

- **zufällige Generierung** der geheimen Ziffernfolge mit Seeding (siehe 2.4.1)
- **korrekte Eingabe** der Ziffern und Steuerbefehle
- **korrekte Verarbeitung** der Eingaben zur benötigten LED-Schaltung
- **zeitnahes Anschalten** der Feedback-LEDs

2.3 Hardware-Setup

Das Herzstück des Systems, mithilfe dessen das Spiel gespielt werden soll, ist der Arduino Mega 2560. Dieser führt alle Berechnungen des aufgespielten Programms durch, liest den Input vom Membrane Switch ein und steuert die LEDs.

Um die Eingaben des Membrane Switch einzulesen, werden dessen Anschlüsse mithilfe von 8 Jumperkabeln mit den digitalen Anschlüssen 2 bis 9 des Arduinos verbunden. Zusätzlich ist es nötig, insgesamt 14 Jumperkabel mit unterschiedlichen Reihen eines Breadboards zu verbinden. Dabei sollte immer mindestens eine leere Steckreihe zwischen allen verwendeten Steckreihen liegen, damit in einem späteren Schritt noch genügend Platz für die LEDs und deren Ground-Kabel bleibt.

Auf dem Breadboard sollte dann jede der unteren Steckreihen, welche mit dem Arduino über ein Jumperkabel verbunden wurde, mithilfe eines Widerstands mit der darüberliegenden Steckreihe mit der gleichen Breadboard-Reihennummer verbunden werden (siehe Abbildung 5). Die Seite starthardware.org empfiehlt dabei 220 Ω Vorwiderstände für die verwendeten 5V LEDs [4].

Diejenigen oberen Steckreihen des Breadboards, welche zuvor mithilfe der Vorwiderstände mit den unteren Steckreihen und damit auch mit den Digitalanschlüssen des Arduinos verbunden wurden, werden nun mit den Anoden der benötigten LEDs versehen. Bei der Anordnung der LEDs ist es wichtig, von links nach rechts jeweils 4 LED-Ampeln zu bilden und abschließend noch 2 weiße LEDs hinzuzufügen. Jede LED-Ampel besteht aus einer grünen LED, einer daneben liegenden gelben LED und einer abschließenden roten LED (siehe Abbildung 5).

Die Kathoden dieser 14 parallel geschalteten LEDs werden in die freien, daneben liegenden oberen Reihen gesteckt und mithilfe von jeweils einem Jumperkabel mit der blau gekennzeichneten oberen Querreihe verbunden. Diese Querreihe wird dann abschließend mithilfe eines letzten Jumperkabels an einen Ground-Input (GND) des Arduinos angeschlossen.

Ein Foto des beschriebenen Aufbaus ist in Abbildung 6 zu sehen. Für den gesamten Systemaufbau werden insgesamt folgende Teile benötigt:

- **1 Arduino** Mega 2560
- **1 Breadboard** mit mindestens 28 parallelen Steckreihen und einer Querreihe
- **1 Membrane Switch 4×4**
- **14 LEDs** (4 grün, 4 gelb, 4 rot, 2 weiß)
- **37 Jumperkabel**
- **14 Widerstände** (220 Ω)

2.4 Implementierung

Im folgenden Abschnitt wird die Entwicklung der Software erläutert, welche die Berechnungen für das Spiel zur Laufzeit auf dem Arduino durchführt. Als Entwicklungsumgebung wird die Arduino IDE in der Version 2.0.1 verwendet. Insgesamt kann der benötigte Code in die beiden Abschnitte Setup und Loop unterschieden werden. Sämtliche Vorgänge innerhalb des Programms können im Codeabschnitt 7 übersichtsweise als Pseudocode eingesehen werden.

2.4.1 Setup-Funktion

Die Setup-Methode eines Arduino-Programms wird jedes Mal aufgerufen, wenn das Programm startet und dient in diesem Projekt der Initialisierung der Keypad-Variablen, der Pin-Modi sowie dem Auslesen eines Seeds zur Initialisierung des Zufallszahlengenerators. Die Abbildung 7 gibt in den Zeilen 1 bis 7 einen Überblick über die wichtigsten Funktionen der Setup-Methode.

Der Input des Membrane Switch, welche innerhalb des Programm-Codes *Keypad* genannt wird, wird mithilfe der Keypad-Library [6] eingelesen. Diese Bibliothek stellt dem Programm Keypad-Funktionen und Datentypen zur Verfügung, um einen simplen Einlesevorgang des Membrane-Switch-Inputs zu gewährleisten. Ausschlaggebend ist hierbei der Aufruf der Keypad-Funktion, welche ein Objekt des Keypad-Datentyps erzeugt. Zu diesem Zweck muss der Keypad-Funktion eine Keymap, die Row-Pin-IDs, die Column-Pin-IDs sowie die Ausmaße des Keypads übergeben werden, wie im Codeabschnitt 8 zu erkennen ist. Mithilfe des erzeugten, globalen Keypad-Objektes besteht dann in jedem Durchlauf der Loop-Methode die Möglichkeit, Input vom Membrane Switch mithilfe der GetKey-Funktion einzulesen.

Die Random-Funktion der Arduino-IDE zur Generierung zufälliger Zahlen verwendet standardmäßig bei jedem Neustart den gleichen Seed. Startet man also einen gewöhnlichen Arduino-Zufallszahlengenerator mehrmals hintereinander und lässt diesen bei jedem Neustart eine zufällige Zahlenfolge generieren, so sind die Zahlenfolgen aus den verschiedenen Durchläufen exakt gleich. Um diese unerwünschte Eigenschaft zu umgehen, wird in der Setup-

Methode dieses Projekts der Zufallszahlengenerator bei jedem Neustart mit einem anderen, zufälligen Seed initialisiert (siehe 7, Zeile 3). Dieser zufällige Seed wird erzeugt, indem das Rauschen an dem unbesetzten analogen Steckplatz A0 des Arduinos eingelesen wird. Ein unbesetzter Steckplatz bietet eine gute Quelle für einen Seed, da diese Art von Steckplatz von elektromagnetischen Feldern aus der Umgebung beeinflusst wird und somit zu jedem Zeitpunkt eine ungewisse, schwankende Spannung vorweist.

Als letzter Schritt der Setup-Methode werden alle Pins mit einer ungeraden ID von 23 bis 49 als Output-Pins zur Verwendung der LEDs initialisiert. Diese Pins wurden aufgrund ihrer räumlichen Nähe beim Systemaufbau ausgewählt, was in Abbildung 5 zu erkennen ist. Die digitalen Pins mit den IDs 2 bis 9 zum Einlesen des Membrane-Switch-Inputs wurden an diesem Punkt bereits innerhalb der Keypad-Funktion über die Parameter *rowPins* und *colPins* initialisiert (siehe Codeabschnitt 8).

2.4.2 Loop-Funktion

Die Loop-Methode stellt den dynamischen Teil eines Arduino-Programms dar und wird so lange immer wieder durchlaufen, wie der Arduino mit Strom versorgt wird. In dieser Methode wird der Spielstatus kontrolliert, es werden Nutzereingaben verarbeitet und es wird das Feedback des Spiels nach jedem Rateversuch berechnet.

Zu Beginn jedes Durchlaufs prüft das Programm, ob im letzten Durchlauf eine Flag für das Zurücksetzen des Spiels oder des aktuellen Rateversuchs gesetzt wurde. Falls ein solcher Fall eintritt, werden entweder sämtliche Variablen und LEDs auf ihren Anfangszustand zurückgesetzt sowie ein neuer Zahlencode generiert, oder es werden die Arrays zur Verarbeitung des aktuellen Rateversuchs geleert. Das Spiel kann dadurch bereits in diesem Schleifendurchlauf mit den durchgeführten Änderungen gespielt werden. Im Anschluss kontrolliert das Programm, ob Flags zur Änderung der Spielstatus-LEDs gesetzt wurden. Über diese Mechanik wird zwischen den beiden möglichen Spielzustandvisualisierungen umgeschaltet sowie die verbleibende Versuchsanzahl durch das Blinken der linken Status-LED angezeigt (siehe Abschnitt 2.1).

Im Anschluss an die Überprüfung des Spielzustands erfolgt das Einlesen des Inputs vom Keypad (siehe Codeabschnitt 7, Zeile 14 bis 21). Zu diesem Zweck muss die GetKey-Funktion des im Setup angelegten Keypad-Objekts aufgerufen und dessen Rückgabewert in einer Char-Variable gespeichert werden. Diese Funktion prüft, ob seit dem letzten Schleifendurchlauf eine Taste auf dem Keypad betätigt wurde und ordnet dieser Tastenposition das gewünschte Zeichen mithilfe der im Setup angegebenen Informationen zu. Falls ein Tastendruck erkannt wurde, so wird überprüft, ob das eingegebene Zeichen eine Ziffer ist. Ist dies nicht der Fall, so muss eine der roten Tasten betätigt worden sein und ein Neustart des Spiels wird durchgeführt (siehe Abschnitt 2.1.1 und Abbildung 3). Ansonsten wird die eingegebene Ziffer im

Rahmen der UpdateGuess-Funktion in das Guess-Array an der nächsten freien Stelle gespeichert. Dies ist ein Char-Array der Größe 4, welches die Ziffern des aktuellen Versuchs festhält.

Den Abschluss der Loop-Funktion bildet die Berechnung und Schaltung der Feedback-LEDs sowie die Überprüfung der Endbedingungen des Spiels (siehe Codeabschnitt 7, Zeile 23 bis 29). Sämtliche Funktionen dieses Abschnitts werden nur ausgeführt, falls das Guess-Array vollständig gefüllt ist und damit der aktuelle Rateversuch komplett eingegeben wurde. Zuallererst wird die Zählvariable guessCount erhöht, welche die Anzahl der durchgeführten Rateversuche dokumentiert. Im Anschluss werden innerhalb der CalcLEDs-Funktion die Berechnungen zur Schaltung der Feedback-LEDs durchgeführt (siehe Codeabschnitt 9).

In diesem Abschnitt werden verschiedene Variablen benötigt, welche im folgenden kurz erklärt werden:

- **UsesOfDigits** ist ein Int-Array der Größe 10 und wird für die Berücksichtigung von Duplikaten unter den Ziffern benötigt. Die Variable speichert, wie oft jede Ziffer von 0 bis 9 bereits mithilfe von LEDs innerhalb eines Rateversuchs dargestellt wird.
- **Occurrences** ist ein Int-Array der Größe 10, in das zu Beginn eines neuen Spiels eingetragen wird, wie oft jede Ziffer von 0 bis 9 in der Spiellösung vorkommt.
- **CurrentLEDs** ist ein Char-Array der Größe 4 und hält fest, welche LED-Farbe an welcher Ziffernposition leuchten soll ('g', 'y' oder 'r' für grün, gelb oder rot).
- **Solution** ist ein Char-Array der Größe 4, in das zu Beginn eines neuen Spiels die generierte Lösung des Spiels eingetragen wird.

Die CalcLEDs-Funktion besteht aus zwei For-Schleifen, deren Laufvariable jeweils über die Indizes des Guess-Arrays (0 bis 3) läuft. In der ersten Schleife (siehe Codeabschnitt 9, Zeile 5 bis 10) wird für jede Ziffernposition überprüft, ob die Ziffer aus dem Guess-Array mit der Ziffer aus dem Solution-Array übereinstimmt. Ist dies der Fall, so wird im CurrentLEDs-Array vermerkt, dass die LED an dieser Position grün leuchten soll. Zusätzlich wird die Zählvariable dieser Ziffer in UsesOfDigits erhöht.

In der zweiten Schleife der CalcLEDs-Funktion (siehe Codeabschnitt 9, Zeile 12 bis 26) wird für jede Ziffer des Rateversuchs festgestellt, ob für diese eine gelbe oder eine rote LED eingeschaltet werden muss. Zuerst wird mithilfe des Occurrences-Arrays für jede Ziffer des aktuellen Rateversuchs überprüft, ob diese mindestens einmal in der Lösung des Spiels vorkommt. Falls dies nicht der Fall ist, kann in das CurrentLEDs-Array an dieser Ziffernposition das Leuchten einer roten LED eingetragen werden. Kommt die Ziffer jedoch in der Lösung vor, dann muss im Anschluss mithilfe des IF-Statements in Zeile 17 ausgeschlossen werden, dass die geratene Ziffer auch an der korrekten Position in der Lösung ist, weil in diesem Fall werder eine gelbe noch eine rote LED eingeschaltet werden darf.

Sobald dieser Fall ausgeschlossen werden kann, muss die aktuell betrachtete Ziffer noch auf Duplikate überprüft werden. Hierfür wird in Zeile 18 durch einen Vergleich der Ziffervorkommen in der Lösung und der Anzahl der bereits fest durch LEDs belegten Ziffern ermittelt, ob genügend Duplikate der geratenen Ziffer in der Lösung vorhanden sind, um noch eine weitere Ziffer dieser Art in diesem Rateversuch mit einer gelben LED anzuzeigen. Ist dies nicht möglich, dann wird für diese Ziffer eine rote LED in CurrentLEDs vermerkt. Andernfalls kann eine gelbe LED vermerkt und die UsesOfDigits für diese Ziffer um eins erhöht werden.

Nachdem für jede Ziffernposition eine LED-Farbe berechnet wurde, kann mithilfe dieser Grundlage in Zeile 26 des Codeabschnitts 7 bestimmt werden, ob das Spiel eine seiner Endbedingungen erreicht hat. Hierfür findet zuerst eine Überprüfung statt, ob alle Plätze des CurrentLEDs-Arrays eine Markierung für eine grüne LED vorweisen. Ist dies der Fall, so wird der Sieg des Spielers als Spielende initialisiert, wodurch alle grünen LEDs für einige Sekunden zum Blinken gebracht werden (siehe Codeabschnitt 10, Zeile 4 bis 19). Der entgegengesetzte Fall tritt im darauf folgenden Codeabschnitt ein, falls genauso viele Rateversuche durchgeführt wurden, wie der Spieler maximal zu Verfügung hat. Hierbei blinken alle roten LEDs für einige Sekunden, um die Niederlage des Spielers anzuzeigen (siehe Codeabschnitt 10, Zeile 22 bis 37).

In dem Fall, dass keine Endbedingung des Spiels erreicht wurde, werden zum Abschluss der Loop-Funktion die LEDs nach den Vorgaben des CurrentLEDs-Arrays in der LightLEDs-Funktion eingeschaltet (siehe Codeabschnitt 7, Zeile 27 und Codeabschnitt 11). Außerdem wird ein Flag gesetzt, welches im nächsten Durchlauf das Zurücksetzen des Rateversuchs auslösen wird.

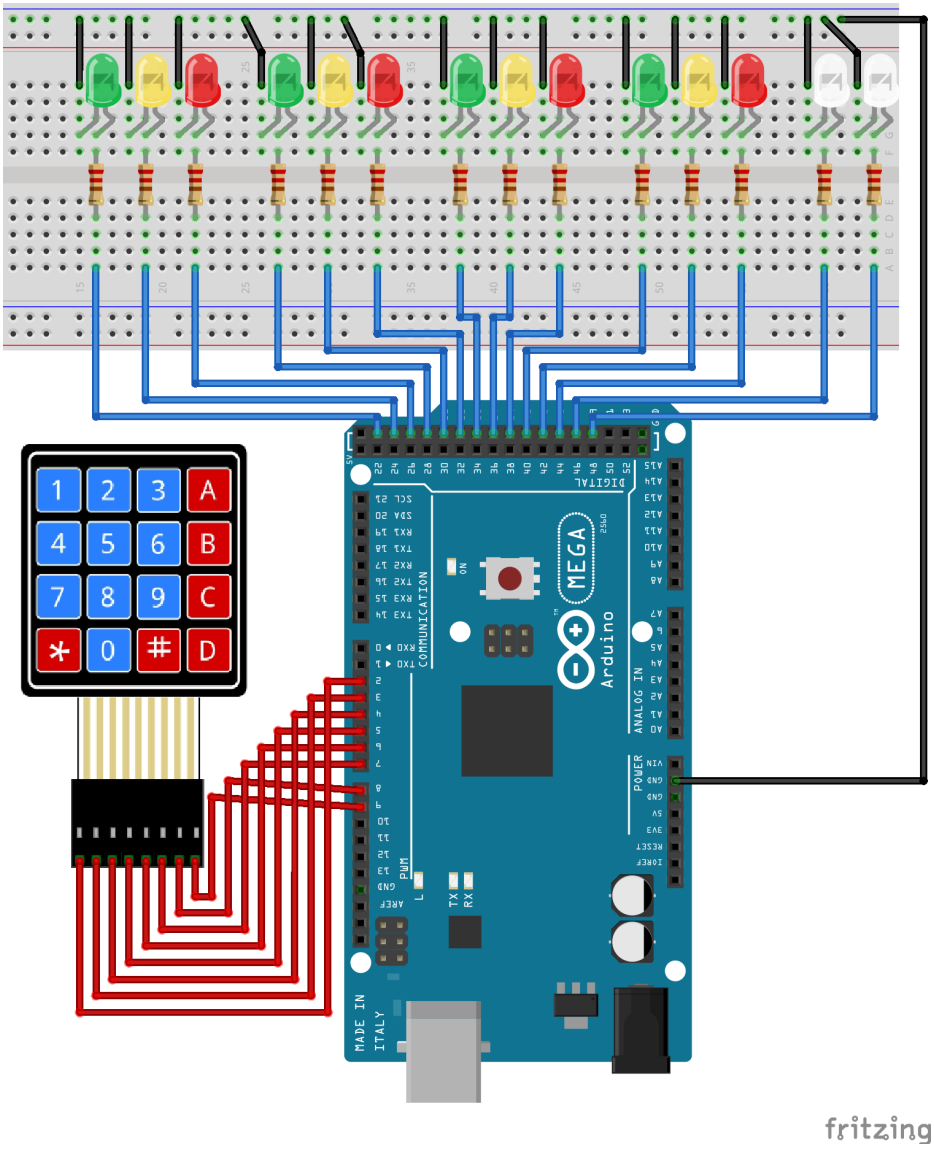

Abbildung 5: Skizze des Systemaufbaus
Quelle: [5]

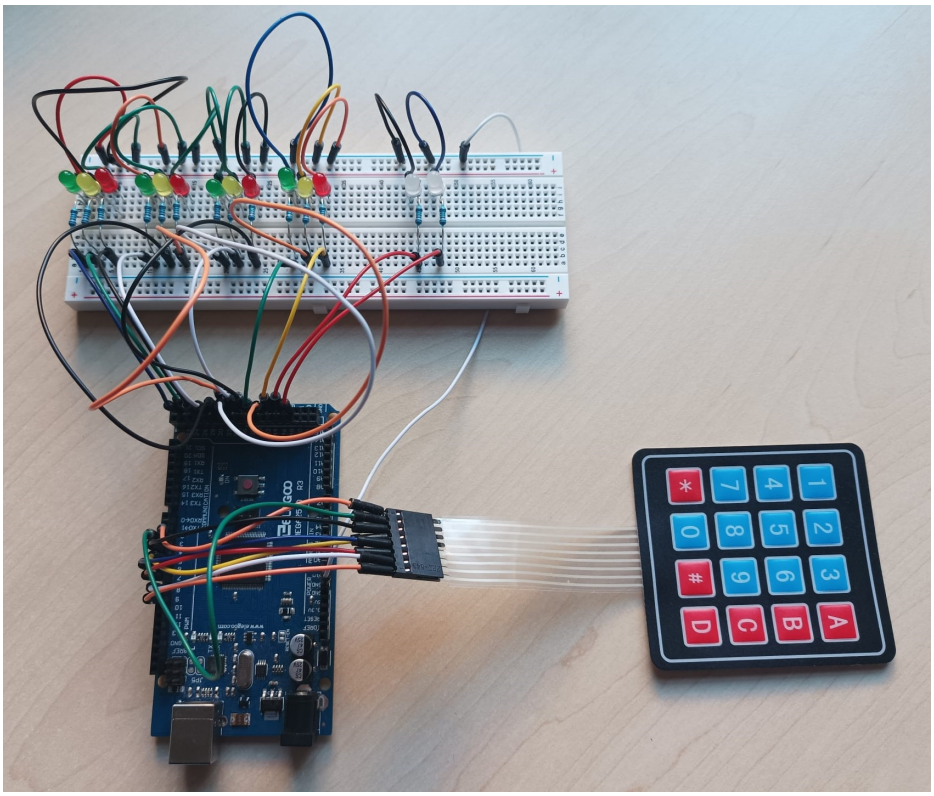

Abbildung 6: Foto des Systemaufbaus
Quelle: eigene Darstellung

```
setup() {
    initKeypad();
    randomSeed(analogRead(A0));
    for (int i = 23; i<=49; i = i+2) {
        pinMode(i,OUTPUT);
    }
}

loop() {
    checkGameReset();
    checkGuessReset();
    controlStatusLEDs();

    char input = customKeypad.getKey();
    if (input) {
        if (isInputDigit(input)) {
            updateGuess(input);
        } else {
            resetGame = true;
        }
    }

    if (isGuessComplete()) {
        guessCount++;
        calcLEDs();
        checkGameEnd();
        lightLEDs();
        resetGuess = true;
    }
}
```

Abbildung 7:
Überblick über die Implementierung in Pseudocode

```
initKeyPad() {
    rowCount = 4;
    colCount = 4;
    hexaKeys = {
      {'1','2','3','A'},
      {'4','5','6','B'},
      {'7','8','9','C'},
      {'*','0','#','D'}
    };
    rowPins = {2,3,4,5};
    colPins = {6,7,8,9};
    customKeypad =
        Keypad(makeKeymap(hexaKeys), rowPins, colPins, rowCount, colCount);
}
```

Abbildung 8:
Code zur Initialisierung des Keypads

```
void calcLEDs() {
  //transform ascii into integer
  int guessedDigit = guess[LEDID]-48;

  for (int i = 0; i < 4; i++) {
    if (guess[i]==solution[i]) {
      currentLEDs[i] = 'g';
      usesOfDigits[guessedDigit]++;
    }
  }

  for (int LEDID = 0; LEDID < 4; LEDID++) {
    int guessOccurrance = occurrences[guessedDigit];
    if (guessOccurrance == 0) {
      currentLEDs[LEDID] = 'r';
    } else {
      if (guess[LEDID] != solution[LEDID]) {
        if (usesOfDigits[guessedDigit] == guessOccurrance) {
          currentLEDs[LEDID] = 'r';
        } else {
          usesOfDigits[guessedDigit]++;
          currentLEDs[LEDID] = 'y';
        }
      }
    }
  }
}
```

Abbildung 9:
Code zur Berechnung der Schaltung der Feedback-LED-Ampeln

```
void checkGameEnd() {
    if (isGreen(currentLEDs[0]) && isGreen(currentLEDs[1])
     && isGreen(currentLEDs[2]) && isGreen(currentLEDs[3])) {
        //win
        for (int i = 23; i<=45; i = i+2) {
                digitalWrite(i, LOW);
        }
        for (int i = 0; i < 6; i++) {
            for (int i = 0; i < 4; i++) {
                digitalWrite(23+i*6, HIGH);
            }
            delay(250);
            for (int i = 0; i < 4; i++) {
                digitalWrite(23+i*6, LOW);
            }
            delay(250);
        }
        resetGame = true;
        return;
    }
    if (guessCount >= maxGuessCount) {
        //lose
        for (int i = 23; i<=45; i = i+2) {
            digitalWrite(i, LOW);
        }
        for (int i = 0; i < 6; i++) {
            for (int i = 0; i < 4; i++) {
                digitalWrite(27+i*6, HIGH);
            }
            delay(250);
            for (int i = 0; i < 4; i++) {
                digitalWrite(27+i*6, LOW);
            }
            delay(250);
        }
        resetGame = true;
        return;
    }
}
```

Abbildung 10:
Code zur Bestimmung, ob eine Endbedingung erreicht wurde

```
void lightLEDs() {
    for (int i = 23; i<=45; i = i+2) {
        digitalWrite(i, LOW);
    }
    for (int i = 0; i < 4; i++) {
        if (currentLEDs[i] == 'g') {
            digitalWrite(23+i*6, HIGH);
        }
        if (currentLEDs[i] == 'y') {
            digitalWrite(25+i*6, HIGH);
        }
        if (currentLEDs[i] == 'r') {
            digitalWrite(27+i*6, HIGH);
        }
    }
}
```

Abbildung 11:
Code zur Schaltung der Feedback-LEDs

3 Evaluation und Fazit

Zusammenfassend kann gesagt werden, dass die Ziele des Projekts erreicht und die geplanten Konzepte erfolgreich umgesetzt wurden. Es konnte ein Logikspiel aus Hard- und Software entwickelt werden, das sich am Konzept des Mastermind-Gesellschaftsspiels orientiert, gleichzeitig jedoch auch eigene Ideen mit technisch-digitalem Bezug zum Arduino umsetzt. Dennoch stehen noch einige Probleme und Verbesserungsmöglichkeiten aus, deren Bearbeitung den Rahmen des Projekts gesprengt hätten.

Eingehend ist zu erwähnen, dass der Membrane Switch dem Nutzer kein ausreichendes Feedback über einen erfolgreichen Tastendruck gibt. Ein haptisches oder auditives Signal bleibt größtenteils aus. Aus diesem Umstand ergibt sich des Öfteren eine Spielsituation, in der der Spieler sich nicht sicher ist, ob ein Tastendruck erfolgreich oder sogar doppelt registriert wurde und dadurch einen Rateversuch verschwenden muss. Das Auswechseln des Membrane Switches gegen ein anderes Modell könnte eine valide Lösung des Problems darstellen, dies ändert jedoch nichts an dem Umstand, dass der Spieler keine Möglichkeit besitzt einzusehen, wie viele und welche Ziffern bereits für den aktuellen Rateversuch eingegeben wurden. Eine wichtige Änderung könnte deshalb umfassen, eine Möglichkeit einzubauen, die aktuelle Eingabe zumindest zu löschen und im besten Fall sogar nachzuvollziehen. Für die Umsetzung dieses Ansatzes könnten weitere LEDs und ein dedizierter Knopf auf dem Membrane Switch verwendet werden.

Ein weiteres Problem des Logikspiels in der vorgestellten Version ist die Übersichtlichkeit und Orientierung der Hardware. Aktuell werden, wie in Abbildung 6 zu erkennen ist, farblich unabgestimmte und teils zu lange Jumperkabel im Systemaufbau verwendet, die aus gewissen Blickwinkeln sogar die Feedback-LEDs verdecken. Dieses Problem ist der Materialbegrenzung des Arduino-Starter-Kits geschuldet und kann leicht durch ein Investment in passendere Materialien gelöst werden. Weiterhin liegt der Membrane Switch in seiner aktuellen Ausrichtung gedreht um 90 Grad nach rechts vor und muss auch so vom Nutzer bedient werden. Dieser Umstand ist der Ausrichtung der digitalen Pins 2 bis 9 des Arduino Mega 2560 geschuldet. Die Anpassung der Jumperkabellängen und Fixierung des Membrane Switches auf einer Unterlage in der korrekten Ausrichtung wäre eine Möglichkeit, mit dem Problem umzugehen. Eine andere Möglichkeit wäre es, digitale Pins in horizontaler Ausrichtung (ab 22 aufwärts, siehe Abbildung 5) für den Membrane Switch zu verwenden. In diesem Fall könnten sich jedoch Breadboard und Membrane Switch überlappen.

Zuletzt fehlt aus der Sicht des Gamedesigns noch eine Spielmechanik, die den Wiederspielwert des Logikspiels erhöht. Denkbar wären hierfür verschiedene Schwierigkeitsgerade mit unterschiedlichen Anzahlen an maximalen Rateversuchen und zeitlichen Beschränkungen für

jeden Rateversuch. Außerdem ist ein Scoring-System denkbar, um die eigene Leistung mit anderen Spielern zu vergleichen.

Abschließend betrachtet hat das Projekt einen gelungenen Einstieg in die Arduino-Entwicklung geboten. Es war eine facettenreiche Betrachtung unterschiedlicher Teilbereiche wie Widerstandsberechnung, Random-Seeding oder auch Character-Mapping mit dem Membrane Switch für die erfolgreiche Umsetzung des Projekts erforderlich. In zukünftigen oder auch aufbauenden Projekt sollte dieses Grundlagenwissen entscheidende qualitative und quantitative Unterschiede in der Bearbeitung ausmachen.

Abbildungsverzeichnis

1 Foto eines vollendeten Mastermind-Spiels Quelle: [2] 2
2 Foto eines elektronischen Mastermind-Spiels Quelle: [2] 3

3 Foto des verwendeten Membrane Switches Quelle: eigene Darstellung 5
4 Foto der LED-Ampeln und Status-LEDs Quelle: eigene Darstellung 6
5 Skizze des Systemaufbaus Quelle: [5] 12
6 Foto des Systemaufbaus Quelle: eigene Darstellung 13
7 Überblick über die Implementierung in Pseudocode 14
8 Code zur Initialisierung des Keypads 14
9 Code zur Berechnung der Schaltung der Feedback-LED-Ampeln 15
10 Code zur Bestimmung, ob eine Endbedingung erreicht wurde 16
11 Code zur Schaltung der Feedback-LEDs 17

Literaturverzeichnis

[1] Nelson, T. *A Brief History of the Master Mind Board Game*. 2000. URL: https://web.archive.org/web/20070812112301/http://www.tnelson.demon.co.uk/mastermind/history.html, besucht am 12. 11. 2022,

[2] Wikipedia. *Mastermind (board game)*. URL: https://en.wikipedia.org/wiki/Mastermind_(board_game), besucht am 13. 11. 2022,

[3] Wikipedia. *Mastermind (Spiel)*. URL: https://de.wikipedia.org/wiki/Mastermind_(Spiel), besucht am 13. 11. 2022,

[4] starthardware.org. *Vorwiderstand für LEDs berechnen*. URL: https://starthardware.org/vorwiderstand-fuer-leds-berechnen/, besucht am 14. 11. 2022,

[5] fritzing.org. URL: https://fritzing.org/, besucht am 14. 11. 2022,

[6] Mark Stanley, A. B. *Keypad*. URL: https://www.arduinolibraries.info/libraries/keypad, besucht am 15. 11. 2022,