Studienprojekt Sommersemester 2023

Dokumentation zu Memory-Math-Quizzer

Ein Projekt von

Gerardo Vasquez

Oussama Lahrnimi

Erik Seiferth

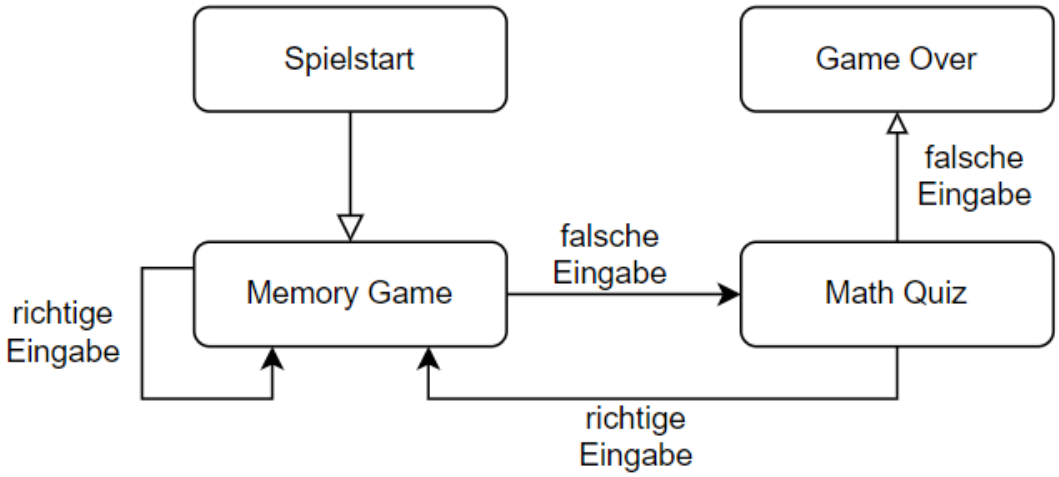
Coburg 06.07.2023

1. Konzept und Idee

Die Idee des Spiels basiert auf einer Kombination aus 2 verschiedenen Spielen:

- Ein Memory Spiel, bei dem der Spieler versuchen muss, sich eine Sequenz an Lichtern zu merken und anschließend durch das Drücken von Knöpfen die Lichter nach simuliert.
- Ein Mathespiel, bei dem der Spieler unter Zeitdruck eine Matheaufgabe auf einem Bildschirm lösen muss und eine Zahl mittels Numpad eintippt.

Der Spielverlauf sieht wie folgt aus:



Das Memory Spiel beinhaltet eine Anzahl von 4 LEDs (rot, grün, gelb, blau). Von diesen leuchten zu Beginn des Spiels 3 Stück nacheinander zufällig auf. Der Spieler ist anschließend an der Reihe durch 4 korrepetierende Knöpfe unterhalb der LEDs diese in der richtigen vorher angezeigten Sequenz zu drücken.

Das Mathe Quiz dient als eine zweite Chance, falls man im Memory Spiel nicht die richtige Reihenfolge gedrückt hat. Hierbei wird dem Spieler auf einem kleinen Display ein Matherätsel gestellt, welches der Spieler unter Zeitdruck lösen muss. Dabei beinhaltet das Matherätsel jeweils nur die Grundrechenarten. Bei der Eingabe der Antwort steht dem Spieler ein Numpad zur Verfügung. Bei richtiger Eingabe kehrt das System zum Memory Spiel zurück, bei falscher Eingabe oder Ablaufen der Zeit ist das Spiel zu Ende und der Bildschirm zeigt ‘Game Over’.

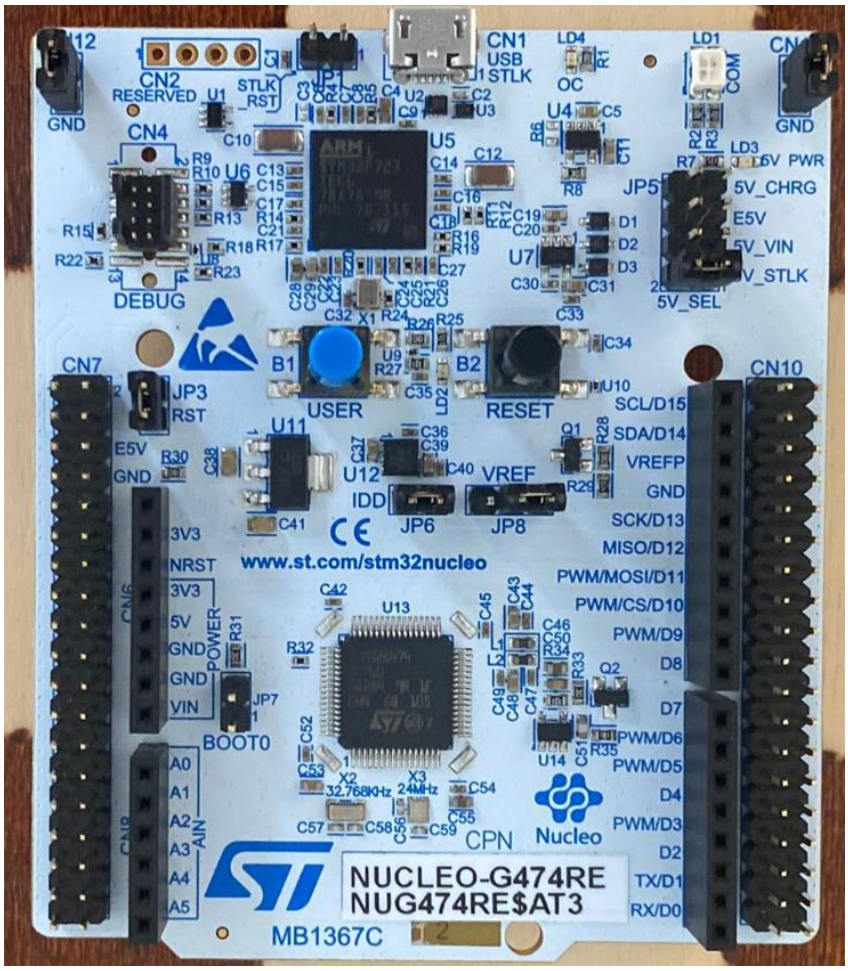
2. Hardware

Die Hardwareliste des Projekts beinhaltet folgende Komponenten:

- Nucleo-G474RE
- LCD-Display ili9341
- Numpad
- 6 LEDs
- 4 Buttons

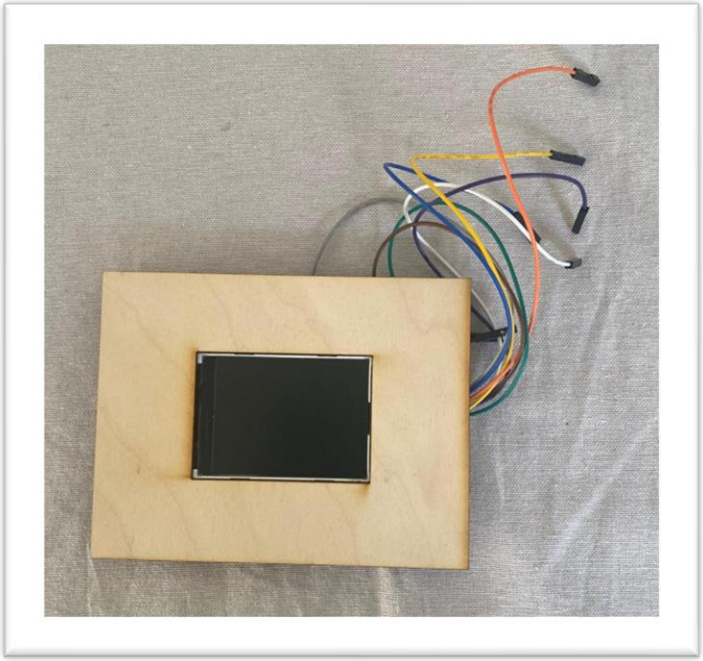
2.1. Nucleo-G474RE

Der Nucleo-G474RE ist ein Entwicklungsboard, welches aufm einem Mikroprozessor basiert.



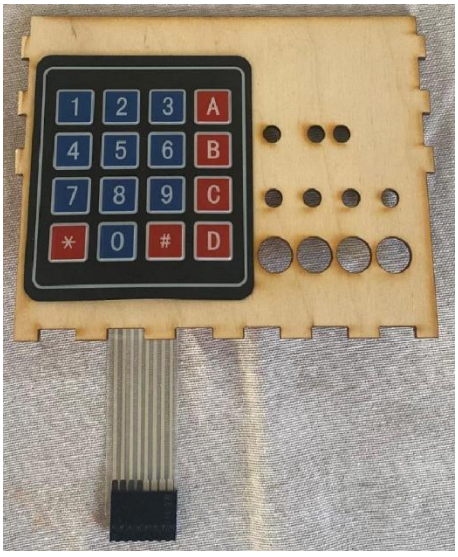
2.2 LCD-Display ILI9341

Das Display dient primär als Anzeige für Informationen. Der Spieler kann darauf seinen derzeitigen Punktestand sehen und den “High-Score”. Der zweite Nutzen ist das Anzeigen der Matherätsel im zweiten Teil des Spiels. Letzlich wird dem Spieler auf der Anzeige noch ein “Game Over” Bildschirm gezeigt, im Falle dass dieser verliert.



2.3 Numpad

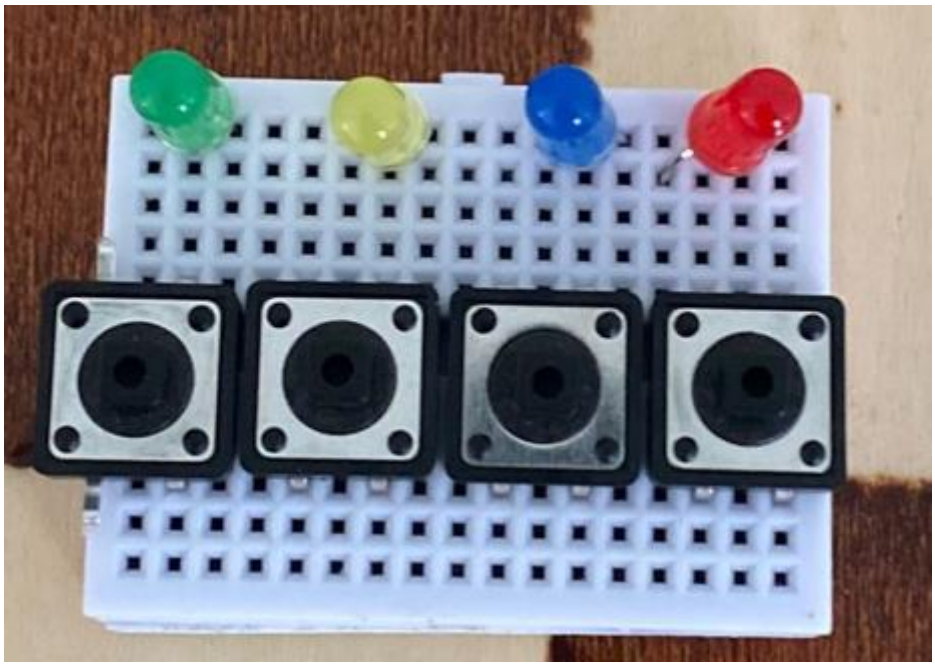
Das Numpad ist eine der Eingabemöglichkeiten des Spielers. Während des Matherätsels ist es dem Spieler dadurch möglch eine Antwort einzugeben.



2.4 LEDs und Buttons

Das Spiel besitzt 6 verschiedene LEDs. 4 dieser LEDs werden für das Memory-Spiel verwendet, um die Sequenz anzuzeigen. Die anderen beiden LEDs (rot und grün) sind Indikatoren für den Spielstatus. Wird z.B. eine Reihenfolge richtig eingetippt blinkt die grüne LED. Ist Sie jedoch falsch blinkt die rote LED 3x mal schnell hintereinander.

Die 4 Buttons befinden sich unter den 4 LEDs für das Memoryspiel. Auf Ihnen wiederholt man die vorgegebene Sequenz des Memoryspiels.



3. Code

Der Code für das Projekt wurde in der IDE Arduino geschrieben. Dadurch startet das Programm in *der .ino* Datei. In dieser legen wir ein Objekt der Klasse LEDGame an. In der *setup()* Funktion wird dann das Spiel gestartet.

```
1   #include <Arduino.h>
2   #include <stdlib.h>
3   #include <LED.h>
4   #include <iostream>
5   #include "Button.h"
6   #include "LEDGame.h"
7
8   LEDGame game;
9   void setup(){
10    game.start();
11  }
```

3.1 LEDGame.cpp

LEDGame.cpp verwaltet den Großteil der Logik des Spiels. Hier lässt sich auch der super-loop finden, welcher zwischen den Zuständen des Spiels schaltet. Unter anderem werden hier auch die Pins für die LEDs gesetzt und Objekte anderer Klassen wie z.B. dem Numpad oder dem Bildschirm erzeugt um diese steuern zu können.

```
17  LED LED1(PC3);
18  LED LED2(PC2);
19  LED LED3(A5);
20  LED LED4(A4);
21  LED TimerLED(A3);
22  LED GreenLED(A2);
23  LED RedLED(A1);
24  std::vector<LED> ledArray = { LED1, LED2, LED3, LED4 };
25  Button Button1(PC10);
26  Button Button2(A0);
27  Button Button3(PC12);
28  Button Button4(PC11);
29  Button startButton(PA13);
30  static uint32_t xorshift_state = 0xABCD1234;
31  int counterLED;
32  Display display;
33  MathQuizzer MathQuiz;
34  Numpad numpad;
```

Anschließend wird zum Starten des Spiels eine Zählvariable gesetzt, welche das derzeitige Level des Memoryspiels angibt. Gleichzeitig kann diese auch als Punktestandvariable verwendet man, was sehr effizient vom Code her. Das Display wird gestartet und mit dem Score und Highscore versehen. Der Spiel-Loop wird nun gestartet.

```
50  void LEDGame::start() {
51      counterLED = 4;
52      display.setup();
53      display.scoreDisplay(999, counterLED - 4);
54      TimerLED.on();
55      bool gameTest = true;
56      while (gameTest == true) {
```

Während *gameTest* true zurückliefert, bleibt das Spiel im GameLoop wobei bei jedem erfolgreichen eintippen der Sequenz die Variable counterLED hochgezählt wird und die Schwierigkeit sich somit auch erhöht.

```
56      while (gameTest == true) {
57
58          std::vector<int> Runde = initRound(counterLED);
59          gameTest = playRound(Runde);
```

```
95          gameTest ? counterLED++ : counterLED += 0;
96          display.scoreDisplay(999, counterLED - 4);
```

Im Falle dass die Sequenz falsch eingegeben wurde, wird der MatheQuiz Teil des Code ausgeführt.

```
if (gameTest == false) {
MathQuiz.createQuiz();
int ergebnis = MathQuiz.getResultado();
bool isMathGame = true;
const char* operation = LEDGame::stringToConstChar(MathQuiz.getOperationString());
display.scoreDisplay(999, counterLED - 4);
display.showMathgame(operation);
delete[] operation;
```

Während das Mathequiz läuft, wird gecheckt ob die Eingabe des Numpads mit dem richtigen Wert des Quiz übereinstimmt. Ist dies der Fall, gelangen wir zurück zum LEDGame und die *gametest* Variable wird wieder auf true gesetzt. Anderenfalls sind beide Variablen auf false und wir gelangen in den Game Over Part.

```
71          if (typed == ergebnis){
72            MathQuiz.setDifficulty(MathQuiz.getDifficulty()+1);
73            isMathGame = false;
74            gameTest = true;
75            delay(300);
76            display.checkmark();
77            delay(300);
78            display.blackScreen();
79          }
80          else{
81            isMathGame = false;
82            gameTest = false;
83            delay(300);
84            display.redX();
85            delay(300);
86            display.blackScreen();
87          }
98        display.gameOver();
99      };
```

3.2 Display.cpp

Die Display.cpp Klasse sorgt für sämtliche Aktionen, die auf dem Display abgespielt werden. Darunter sind die wichtigsten das Anzeigen der Punktestände, das Anzeigen von Fehler/Richtig Animationen und das Anzeigen des Mathequiz.

```
61  void Display::showMathgame(const char* mathgame){
62    tft.fillRect(30, 80, 25, 25, ILI9341_BLACK);
63    tft.setTextSize(5);
64    tft.setCursor(60, 70);
65    tft.println(mathgame);
66  }
```

```
33  void Display::scoreDisplay(int highscore, int score){
34      // Draw the horizontal line at 20% of the screen height
35    int lineY = tft.height() * 0.1;
36    tft.drawFastHLine(0, lineY, tft.width(), ILI9341_WHITE);
37
38    // Draw the "score:" label at the top left corner
39    tft.setTextSize(2);
40    tft.setCursor(0, 0);
41    tft.print("Score:");
42    tft.print(score);
43
44    // Draw the "highscore:" label at the top left corner
45    tft.setTextSize(2);
46    tft.setCursor(160, 0);
47    tft.print("HighScore:");
48    tft.print(highscore);
49  }
```

```
87  void Display::redX(){
88      tft.fillScreen(ILI9341_BLACK);
89      tft.drawLine(0, 0, tft.width(), tft.height(), ILI9341_RED);
90      tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_RED);
91      tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_RED);
92    }
93
94  void Display::blackScreen(){
95      tft.fillScreen(ILI9341_BLACK);
96    }
97
98  void Display::checkmark(){
99      tft.fillScreen(ILI9341_BLACK);
100     int checkmarkWidth = tft.width() * 2 / 3;
101     int checkmarkHeight = tft.height() * 2 / 3;
102     int checkmarkX = (tft.width() - checkmarkWidth) / 2;
103     int checkmarkY = (tft.height() - checkmarkHeight) / 2;
```

3.3 Numpad.cpp

Die Numpad.cpp Klasse besitzt Funktionen für die Eingabe mit dem Numpad. Diese wird als Matrix ausgelesen und muss dementsprechend konfiguriert werden.

```
4   const byte ROWS = 4; //four rows
5   const byte COLS = 4; //four columns
6   //define the cymbols on the buttons of the keypads
7   char hexaKeys[ROWS][COLS] = {
8     {'1', '3', '2', 'A'},
9     {'4', '6', '5', 'B'},
10    {'7', '9', '8', 'C'},
11    {'*', '#', '0', 'D'}
12  };
```

```
26  char Numpad::getNumber(){
27    while(true){
28    char customKey = customKeypad.getKey();
29
30    if (customKey){
31      Serial.println(customKey);
32      return customKey;
33      }
34    }
35  };
```

4. Quellen

Arduino IDE – Entwicklungsumgebung - https://www.arduino.cc/en/software/

Adafruit ILI9341 – Grafik Library für das Display - https://github.com/adafruit/Adafruit_ILI9341

Keypad by Mark Stanley – Library für das Numpad - https://playground.arduino.cc/Code/Keypad/