Erweiterte Aufgaben

Einführung

Sie haben sich nun intensiv mit dem Projekt Vier-Gewinnt auseinandergesetzt. Dabei lag der Fokus zum größten Teil auf der Hardware (dem technischen Teil). Mit den folgenden Aufgaben sollen sie sich näher mit dem Programm und der Spiellogik vertraut machen. Dafür haben wir Ihnen ein Grundgerüst zur Verfügung gestellt mit dem Sie arbeiten können. Damit sie sich in dem gegebenen Code zurechtfinden und diesen erweitern können folgen einige Erläuterungen. Außerdem existiert eine „Musterlösung“, welche Sie gerne zur Hilfestellung verwenden können. Seien Sie allerdings ehrgeizig und geben Sie nicht zu früh auf, durch eigenes Denken lernen Sie mehr als durch Abschreiben 😉.

Begriffserklärung

In den folgenden Aufgaben werden einige Fachbegriffe verwendet. Die häufigsten werden im Folgenden einmalig erklärt und dann im weiteren Verlauf als gegeben vorausgesetzt. Falls die Erklärungen nicht beim Verständnis weiterhelfen, lohnt es sich auch die Begriffe im Internet nachzuschlagen. Des Weiteren wird in diesem Dokument zur Beschreibung der Syntax ein Format verwendet, bei dem Spitzklammern (<>) wie Platzhalter behandelt werden. Diese sollen nicht im Code übernommen werden, sondern müssen durch passende Namen oder ähnliches ersetzt werden.

Beispiel:

lm.<Methodenname>(<Parameterliste>)

kann durch

lm.placeStone(2, Color::GREEN);

ersetzt werden.

Variable: Eine Variable ist eine Art Platzhalter für einen Wert. So kann man Werte speichern und anpassen. Der Zugriff auf diese Werte erfolgt durch den Namen der Variable. In einer Variable können nur Werte des gleichen Typs gespeichert werden. Für eine Liste der verschiedenen Typen können Sie die Arduino Reference verwenden, denn dort befindet sich ein Unterpunkt namens „Variables“: https://docs.arduino.cc/language-reference/de/

Methode: Um ein Programm strukturierter zu gestalten verwendet man sogenannte Methoden (oder auch Funktionen). In sie kann man verschiedene Befehle reinschreiben, die häufiger verwendet werden. Sie besitzen ebenfalls einen Namen und können durch diesen aufgerufen werden. Außerdem kann man

einer Methode Parameter übergeben. Diese funktionieren wie Variablen, können allerdings nur in der Methode verwendet werden.

Array: Ein Array ist eine Variable in der mehr als nur ein Wert gespeichert werden kann. Die Werte haben eine Reihenfolge/ Ordnung, durch diese kann man steuern auf welchen Wert man zugreifen kann. Ein Array kann mehrere Dimensionen besitzen. Für unseren Fall sind es Arrays mit zwei Dimensionen. Diese können Sie sich vorstellen, wie eine Art Koordinatensystem, wo man durch zwei Koordinaten einen genauen Punkt/ Wert identifizieren kann. Dieses Prinzip kennen Sie eventuell auch durch das Spiel „Schiffe versenken“.

Main

Die Main-Datei hat zwei Aufgaben: Zum einen wird hier die Pinbelegung festgelegt und im Setup richtig konfiguriert und zum anderen wird hier die Spiellogik implementiert und geregelt. Für Sie relevant sind dabei nur die Methoden *loop()* und *readButtons()*, da der Rest bereits so vorbereitet wurde, dass es mit der von Ihnen fertiggestellten Platine kompatibel ist. Neben den beiden Methoden werden Sie für die Erweiterungen auch einige der Variablen brauchen, die oben initialisiert wurden. Dazu werden Sie allerdings passende Hinweise bei den Aufgabenbeschreibungen finden.

LEDMatrix und Color

Die Datei mit dem Namen LEDMatrix.cpp ist dafür da, dass man das sichtbare Spielfeld (also die LEDs) ansteuern und verändern kann. Auch hier wurden Ihnen bereits einige Methoden zur Unterstützung bereitgestellt. Hier sind nur die Methoden *setLightValue()*, *placeStone()* **und** *printNumber()* für Sie interessant.
Die Datei mit dem Namen Color.h ist ein sogenannter Enum, welcher unsere Farblogik regelt. Enums werden dafür verwendet, um feste Zustände abzubilden, so wie bei uns die Farben einer LED. Dabei unterscheiden wir zwischen den Zuständen *OFF, RED, GREEN, FLASH_RED und FLASH_GREEN*. Auf diese Zustände kann man zugreifen, indem man folgende Syntax verwendet: *Color::<Zustand>*

Aufgabe 1 – Nummern anzeigen

Wie Sie bereits gesehen haben, hat das Vier-Gewinnt Spiel ein Menü, in dem ein Spielmodus ausgewählt werden kann. Die Spielmodi bilden wir durch Nummern ab (eine Eins für den Einspieler und eine Zwei für den Zweispieler Modus). Die Methode, welche die Nummern auf der LED-Matrix anzeigt haben wir Ihnen breitgestellt (*printNumber()*). Damit Sie besser verstehen, wie der Zustand der LEDs im Programm geändert werden kann, sollen Sie sich die Methode genauer anschauen und erweitern. Die Erweiterung soll dafür sorgen, dass nicht nur die Nummern 1 und 2 angezeigt werden können, sondern alle Nummern von 0-4. Außerdem soll die *loop()-Methode* in der Main-Datei ebenfalls so

angepasst werden, dass die Nummern in einem Zyklus auf den LEDs abgebildet wird. Achten Sie darauf, dass Sie Delays mit einbauen, damit die Nummern lange genug angezeigt werden. Dabei kann die Variable *currentColumn* angepasst und als Parameter an die *printNumber()-Methode* übergeben werden.

Hinweise

Um einen Zyklus zu realisieren, genügt es nicht den Wert immer wieder zu erhöhen. Es muss einen Kipppunkt geben, an dem der Wert wieder zurückgesetzt wird. Dies lässt sich leicht mit dem sogenannten Modulo-Operator lösen. Dieser gibt den Restbetrag einer ganzzahligen Division zurück. Ein Beispiel: $15 \div 6 = 2\ Rest\ 3$. Also würde der Modulo-Operator den Wert 3 berechnen. So haben Sie wahrscheinlich auch die Division in der Grundschule kennengelernt, bevor sie mit Dezimalzahlen gerechnet haben. Im Programm schreiben Sie für den Modulo-Operator ein Prozentzeichen (%), also 15 % 6. Mit der Rechnung:

```
currentColumn = (currentColumn + 1) % 5;
```

kriegen Sie den gewünschten Zyklus von 0 bis 4. Überprüfen Sie die Rechnung gerne mit einigen Werten, um diese nachzuvollziehen.

In der Methode *printNumber()* wird ein *Switch-Case* verwenden. Dies ist eine abgewandelte Form einer *If-Abfrage*, welche in bestimmten Fällen besser zu lesen ist. Damit Sie diese verstehen und anpassen können finden Sie hier einen Link zur Dokumentation von *Switch-Cases*: https://docs.arduino.cc/language-reference/de/struktur/control-structure/switchCase/

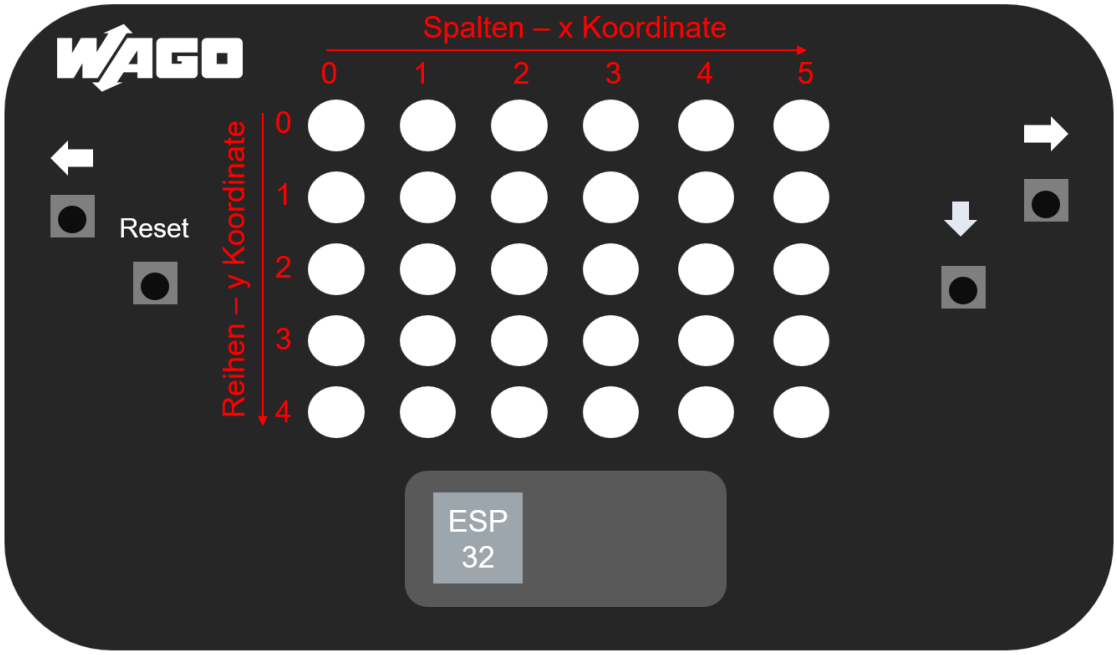
Aufgabe 2 – Auswahlleiste

Vor jedem Spielzug befindet sich der Spieler bei unseren Vier-Gewinnt in einer Art Auswahlleiste (In der man die Spalte anwählen kann, wo der Stein platziert werden soll). In dieser kann man sich mit den Tastern bewegen, ohne das Spiel zu beeinflussen. Erst wenn der Taster nach unten betätigt wird, wird der Stein auch wirklich platziert. Diese Auswahlleiste sollen Sie nun Implementieren. Um dies zu realisieren müssen Sie die Method *readButtons()* in der Main-Datei und die Methode *setLightValues()* in der LEDMatrix.cpp anpassen.

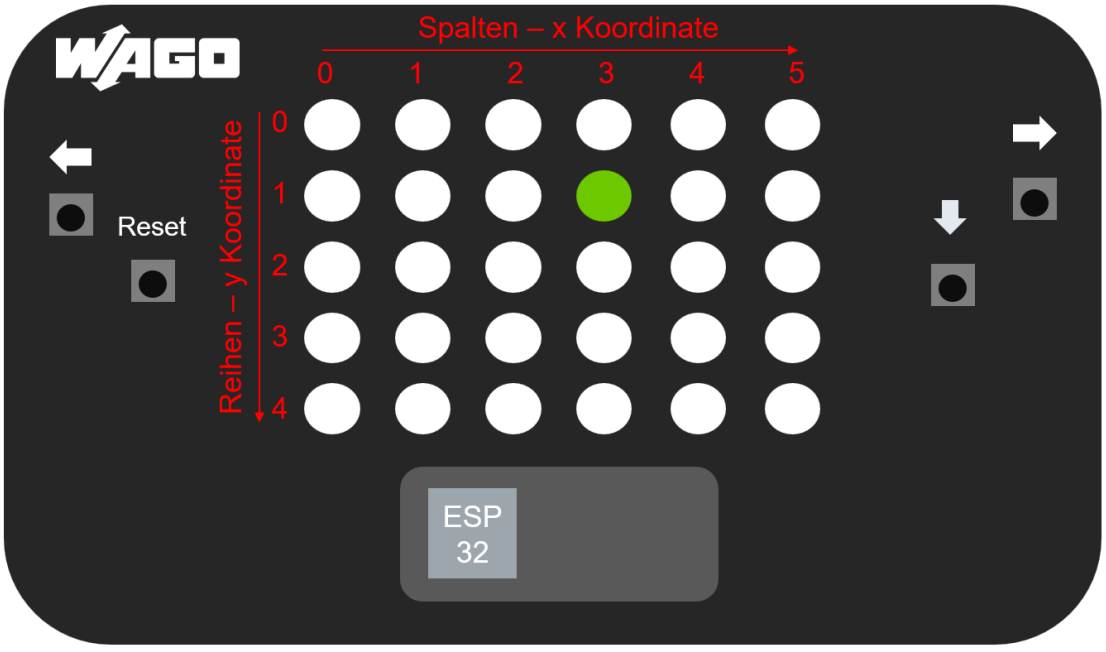
Beginnen Sie mit der Methode *setLightValues()*. Diese erhält drei Parameter: die aktuelle Spalte, die eingeschaltet wird, die vorherige Spalte und die Farbe, die die LED in der Auswahlleiste annehmen soll. Damit die Änderungen auf dem Spielfeld angezeigt werden muss der Zweidimensionale Array *LEDValues* angepasst werden, wie Sie es bereits in der *printNumbe()* Methode getan haben. Allerdings gibt es einen leichteren Weg, um einzelne Werte zu verändern. Mit folgender Syntax greifen Sie auf einzelne Werte des Arrays zu:

```
LEDValues[<x-Koordinate>] [<y-Koordinate>];
```

Dabei ist die LED oben links die LED mit den Koordinaten (0;0). Die Werte, die also möglich sind, liegen für die x-Koordinate zwischen 0 und Anzahl Spalten – 1 und für die y-Koordinaten zwischen 0 und Anzahl Reihen – 1.



Der Befehl *LEDValues[3] [1]* ***greift also auf folgende LED zu:***



Da wir uns in der obersten Reihe befinden bleibt die y-Koordinate für die auswahlleiste immer 0. Wenn Sie sich vorstellen, dass Sie den Stein (LED) mit den Tastern in dieser Auswahlleiste navigieren wollen, müssen Sie dafür die LED, die vorher geleuchtet hat

ausschalten und die neue LED muss in der übergebenen Farbe aufleuchten. Die Farbe, die übergeben werden muss, ist:

Color::FLASH_GREEN,

damit die LED am Ende auch blinkt, wie im richtigen Spiel. Versuchen Sie dies im Programmcode umzusetzen

Nun bearbeiten wir die Methode *readButtons()*. Hier werden die Taster eingelesen und auf das gegebene Eingabe Signal reagiert. Um die Taster einzulesen kann die Methode *digitalRead()* verwendet werden. Diese erhält als Parameter den Pin, welcher gelesen werden soll und gibt einen digitalen Wert (also 0/LOW oder 1/HIGH) zurück. Mit der Abfrage

if (digitalRead(buttonL))

wird abgefragt, ob der Taster links gedrückt wurde. Dies können Sie für den rechten Taster identisch machen. Da keine zwei Taster gleichzeitig gedrückt werden sollen, kann man hier *if…else if* verwende und keine zwei einzelnen If-Anweisungen. Wenn der linke Taster gedrückt wurde, muss die aktuelle Spalte um ein reduziert werden und für den rechten Taster um eins erhöht. Allerdings muss hier berücksichtigt werden, dass das Spielfeld links und rechts einen Rand hat. Hier können Sie wieder die Formel für den Zyklus verwenden. Also*:*

currentColumn = (currentColumn + 1) % nColumns

für den rechten Taster und

currentColumn = (currentColumn – 1 + nColumns) % nColumns

für den linken. Das zusätzliche Aufaddieren bei der Formel für den Linken Taster ist dafür da, um negative Werte zu vermeiden. Bevor Sie die aktuelle Spalte anpassen müssen Sie sich den aktuellen Wert in einer Hilfs-Variable abspeichern, da Sie diesen Wert als Parameter an die Methode *setLightValue()* übergeben müssen. Nun haben Sie alle Berechnungen durchgeführt und können Ihre Werte an die Methode übergeben, dabei würde ich Ihnen empfehlen den Methodenaufruf am Ende außerhalb der *If-Anweisungen* durchzuführen. Da die Methode in einer anderen Datei implementiert ist, müssen Sie die Methode mit

lm.<Methodenname>(<Parameterliste>)

aufrufen. Testen Sie Ihr Programm.

Wie Ihnen beim Testen mit Sicherheit aufgefallen ist, führt das einmalige Betätigen des Tasters zu mehreren Verschiebungen in der Auswahlleiste. Dies liegt daran, dass die *readButtons()-Methode* sehr häufig und schnell hintereinander in der *loop()-Methode* aufgerufen wird. Deshalb kann man nicht so schnell drücken, wie die Methode aufgerufen wird. Aus diesem Grund kann man die Hilfsvariable *lastButton* nutzen. Diese Speichert den Pin des letzten Tasters. Wir nun ein Taster gedrückt, dann passt man diese Variable auf den aktuellen Taster an, zum Beispiel*:*

lastButton = buttonL.

Nun kann man die If-Anweisung erweitern und sagen, dass die aktuelle Spalte nicht nur angepasst werden soll, wenn ein Taster gedrückt wird, sondern auch wenn dieser vorher nicht gedrückt wurde. Des Weiteren muss ein *else-Zweig* hinzugefügt werden, der die *lastButton* Variable wieder zurücksetzt, wenn kein Taster mehr gedrückt wird. Nun sollte Ihre Auswahlleiste richtig funktionieren.

Aufgabe 3 – Steinplatzieren

Zum Schluss wollen wir uns nicht nur in der Auswahlleiste bewegen, sondern auch wirklich Steine platzieren. Dafür müssen zwei Änderungen vorgenommen werden: Die *readButtons()* Methode muss erweitert werden, so dass auch der Taster nach unten erkannt wird und es muss die Methode *placeStone()* in LEDMatrix.cpp implementiert werden.

placeStone() erhält als Parameter die Spalte, in der der Stein platziert werden soll und die Farbe. Um nach der richtigen Stelle in der angegebenen Spalte zu Suchen empfiehlt sich eine For-Schleife zu verwenden. Eine Beschreibung dieser Schleife finden Sie hier: https://docs.arduino.cc/language-reference/de/struktur/control-structure/for/. Eine *For-Schleife* ist eine Zählerschleife. Mit dieser können Sie die Zeilen in der übergebenen Spalte nacheinander überprüfen. Dabei sollten Sie unten beginnen, da der Stein bei einem echten Vier-Gewinnt Spiel durch die Gravitation nach unten fallen würde. Sobald Sie eine LED finden, die ausgeschaltete ist können Sie den Stein dort Platzieren, indem Sie die LED auf den übergebenen Wert setzten. Danach können Sie die Methode mit dem Schlüsselwort *return* beenden.

Für den Taster nach unten können Sie so vorgehen wie für die anderen Taster. Allerdings wird hier die Methode *placeStone()* aufgerufen und nach dem Aufruf wird die aktuelle Spalte auf 0 gesetzt.