Escape-Würfel	**Teil 2**
Ein erstes Gesamtprogramm	

1. Das Hauptprogramm und die Idee des Top-Down-Programmierens

   Das Hauptprogramm für den Escapewürfel kann erstaunlich kurzgehalten und auch schnell verstanden werden:

```
bool geloest = 0;

void loop() {
  neuer_wert();
  wert_hinzufuegen();
  loesungs_check();
  if(geloest==1) code_ausgabe();
}
```

   Das Programm wartet zunächst darauf, dass sich die Lage des Würfels ändert und hält diesen neuen Wert fest. Er wird der Sequenz der letzten Lageinformationen hinzugefügt. Anschließend wird überprüft, ob diese Sequenz mit der vorgegebenen Lösungssequenz übereinstimmt. Ist das der Fall, dann soll der Lösungscode ausgegeben werden. Andernfalls wird wieder an den Anfang der Hauptschleife zurückgesprungen und erneut auf den nächsten neuen Wert gewartet.

   In dieser Form ist das Programm noch lange nicht lauffähig, denn die hier aufgerufenen Unterprogramme existieren ja bisher gar nicht. Es handelt sich vielmehr um eine grobe Lösungsstrategie, d.h. der Lösungsalgorithmus für das Problem wurde eigentlich bisher nur in eine Abfolge sinnvoller Abschnitte zerlegt. Diese Abschnitte müssen in der Folge konkret bis in die Details ausformuliert werden. Dabei kann es eventuell sinnvoll sein, sie ebenfalls in mehrere Unterabschnitte zu zerlegen, die Unterabschnitte eventuell ebenfalls, usw. Das lässt sich bis auf eine unterste Ebene fortsetzen, auf der dann alle Details konkret berücksichtigt und umgesetzt sind. Diese Herangehensweise wird Top-Down-Methode genannt.

   Noch eine Bemerkung zur Variablen „geloest“: Es handelt sich um eine „Boolean-Variable“, die nur zwei Werte annehmen kann, nämlich „wahr“ bzw. 1 oder „falsch“ bzw. 0.

2. Einen neuen Wert auslesen; außerdem globale und lokale Variablen

   Der erste Abschnitt, der in diesem Sinne ausformuliert werden soll, ist das Unterprogramm in dem einer neuer Wert ausgelesen wird:

```
String aktuell = "z up";

void neuer_wert(){
  String neu = myAcc.getOrientationAsString();
  while(neu == aktuell){
    delay(100);
    neu = myAcc.getOrientationAsString();
  }
  delay(500);
  aktuell = myAcc.getOrientationAsString();
}
```

In einer Stringvariablen mit dem Namen „aktuell“ ist die aktuelle Ausrichtung des Würfels hinterlegt. Wird diese Ausrichtung geändert, dann soll die neue Ausrichtung zunächst als neuer Wert festgehalten werden. Dazu wird in diesem Unterprogramm permanent der Lagesensor ausgelesen und mit der bisherigen Lage verglichen. Das geschieht in der while-Schleife, die erst dann verlassen wird, wenn sich der ausgelesene Wert und der bisherige Wert unterscheiden. Dann wird kurz gewartet, denn während der Rotation des Würfels sind die Lagewerte eine kurze Zeit nicht stabil. Im Anschluss wird der neue aktuelle Wert ausgelesen und in der Variablen „aktuell“ hinterlegt.
Interessant ist ein Unterschied zwischen den beiden String-Variablen „aktuell“ und „neu“. Die Variable „aktuell“ wird vor allen folgenden Programmteilen deklariert und ist damit immer und aus allen Programmteilen heraus verfügbar. Man nennt Variablen dieser Arte „globale Variablen“. Das klingt zunächst mal sehr komfortabel, wenn man sich keine Gedanken darüber machen muss, ob man die Variable an einer bestimmten Programmstelle nutzen kann. Es hat aber den Nachteil, dass für diese Variable auch permanent Speicherplatz reserviert sein muss. Deshalb zielt guter Programmierstil darauf ab, Variablen möglichst nur in den Bereichen festzulegen, wo sie wirklich gebraucht werden, und den Speicherplatz dann wieder freizugeben. Das wird hier mit der Variablen „neu“ praktiziert. Sie wird nur innerhalb des Unterprogramms „neuer Wert“ benötigt und deshalb auch innerhalb diesen Unterprogramms deklariert. Bei Verlassen des Unterprogramms ist dagegen nicht mehr existent. Eine solche Variable wird „lokale Variable“ genannt. Wenn man zusätzlich dafür sorgt, dass sich Unterprogramme Werte übergeben, lassen sich viele globale Variable elegant vermeiden.

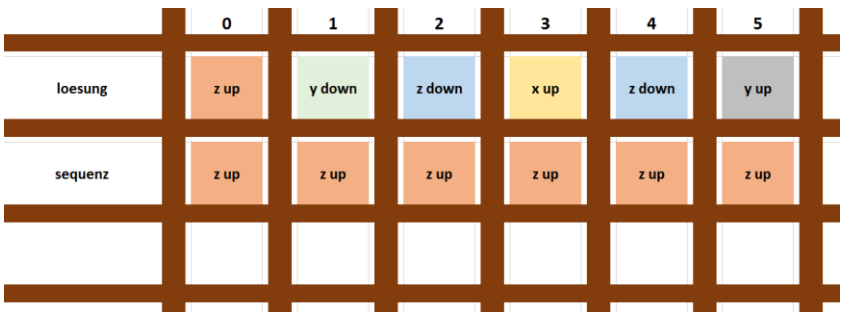
3. Arrays

Die gewünschte Lösung besteht beim Escape-Würfel nicht in einer bestimmten Würfellage und einem entsprechenden Wert, sondern es geht um eine Abfolge von Lagen, die nacheinander eingenommen werden sollen. Benötigt werden also mehrere Variablen. Jede soll eine Würfelausrichtung speichern, wobei die zeitliche Reihenfolge der Lagen in einer sinnvollen Organisation der Variablen aufgegriffen werden muss.
Das leisten sogenannte „Arrays“:

```
String aktuell = "z up";

String loesung[]={"z up", "y down","z down","x up","z down","y up"};
String sequenz[]={"z up", "z up", "z up", "z up", "z up", "z up"};
```

Im Vergleich zur Variablen „aktuell“, die einen String aufnehmen kann, nimmt das Array „loesung[]“ in diesem Fall sechs Strings auf. Man kann sich ein solches Array wie ein Regal vorstellen, in das die einzelnen Variablen-Kartons eingestellt werden können:



Die Werte der Variablen lassen sich setzen oder auslesen, indem man den Ort der Ablage im Regal in den eckigen Klammern hinzufügt.

Beispiel: loesung[3] ist die im Regal in der loesung-Etage an Platz 3 dargestellte gelbliche Variable, die den String „x up“ beinhaltet.

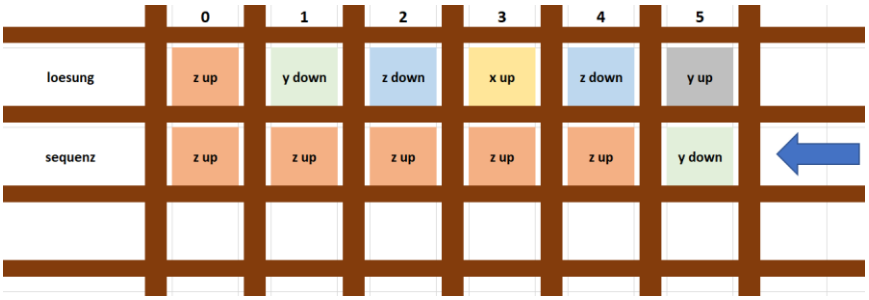
Für unser Programm kommen sogar zwei Arrays zum Einsatz: Im Array „loesung“ wird die gewünschte Abfolge von Würfelausrichtungen festgelegt, die im weiteren Verlauf nicht mehr verändert wird. Im Array „sequenz“ werden die letzten 6 Würfelausrichtungen festgehalten, die der Würfel nacheinander hatte. Die zu Beginn als Anfangswerte jeweils mit „z up“ belegten einzelnen Variablen müssen nach und nach durch wirkliche Messwerte überschrieben werden.

4. <u>Einen neuen Wert hinzufügen und die Idee des Verschiebens von Werten innerhalb eines Arrays</u>

Wurde ein neuer Wert für die Lage des Würfels ausgelesen, dann muss er der bisherigen Sequenz hinzugefügt werden. Das leistet das folgende Unterprogramm:

```
void wert_hinzufuegen(){
  for(int i=0; i<5; i=i+1){
    sequenz[i]=sequenz[i+1];
  }
  sequenz[5]=aktuell;
}
```

Das lässt sich im Regalbild gut nachvollziehen:



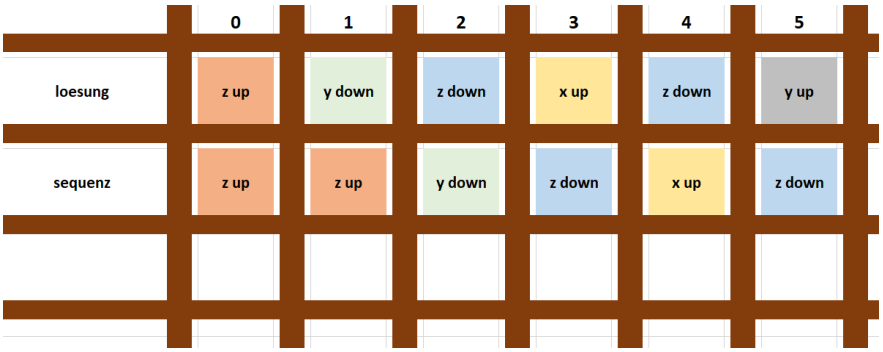
Wird als neuer Wert beispielsweise „y down“ ausgelesen, dann werden (in der for-Schleife) zunächst alle Kartons auf dem sequenz-Regalbrett um einen Platz nach links geschoben. Der ganz linke Karton fällt dabei aus dem Array heraus. In die rechts entstehende Lücke wird dann der neue Wert „y down“ neu eingestellt.

5. <u>Vergleich der aktuellen Sequenz mit der Lösungssequenz</u>

Nach und nach werden viele Würfelausrichtungen festgehalten und hoffentlich ist irgendwann auch mal die richtige Abfolge dabei. Das muss natürlich nach jedem Hinzufügen eines neuen Wertes überprüft werden, was das folgende Unterprogramm leistet:

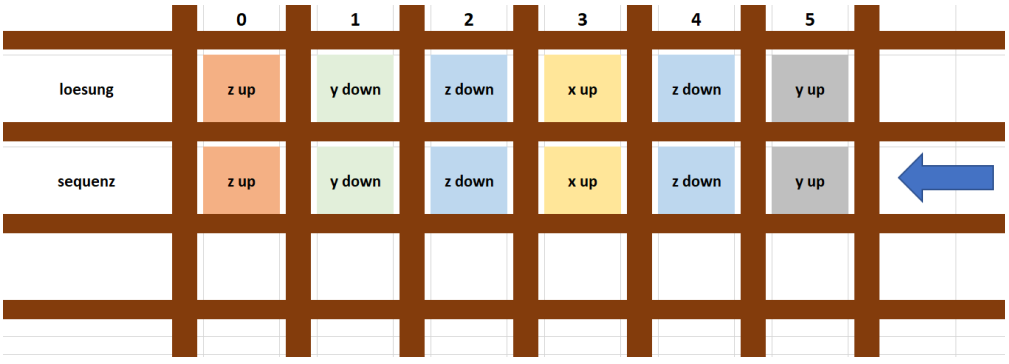
```
void loesungs_check(){
  Serial.print("soll: ");
  for(int i=0; i<6; i=i+1){
    Serial.print(loesung[i]);
    Serial.print("   ");
  }
  Serial.println(" ");
  geloest = 1;
  Serial.print(" ist: ");
  for(int i=0; i<6; i=i+1){
    Serial.print(sequenz[i]);
    Serial.print("   ");
    if(sequenz[i]!=loesung[i]) geloest = 0;
  }
  Serial.println(" ");
  Serial.println(" ");
}
```

Zunächst fallen viele Ausgabezeilen auf den seriellen Monitor auf, die den Komfort der Programmnutzung erhöhen sollen, den Code aber natürlich umfangreicher machen. Für das generelle Verständnis sind sie unerheblich. Dafür hilft wieder das Regalbild:



Es wird zunächst durch setzen der Variablen „geloest“ auf 1 so getan, also ob die Sequenz die Lösungsreihenfolge erfüllt. Dann wird Regalspalte für Regalspalte überprüft, ob die zwei übereinander angeordneten Kartons übereinstimmen. Stimmt das auch nur in einer Spalte nicht, dann wird die Variable „geloest“ auf 0 korrigiert. Im Bild oben erfüllt die Spalte ganz links die Anforderung, bereits in der Spalte daneben rechts scheitert der Vergleich aber schon.

Wird als nächster Wert „y up“ ausgelesen, hat der Spieler aber gewonnen:



Jetzt wird jede einzelne Spaltenüberprüfung das Ergebnis liefern, dass die beiden Einträge übereinstimmen. Somit bleibt der Wert der Variablen „geloest“ auf 1 und im Hauptprogramm wird die Ausgabe des Lösungscodes aufgerufen.

6. Die Ausgabe des Lösungscodes

   In diesem Fall wird der Lösungscode ausgegeben:

```
void code_ausgabe(){
  Serial.println("Geschafft!");
  for(int i=0; i<blau; i=+1){
  digitalWrite(blauPin, HIGH);
  delay(500);
  digitalWrite(blauPin, LOW);
  delay(500);
 }
  for(int i=0; i<gelb; i=+1){
  digitalWrite(gelbPin, HIGH);
  delay(500);
  digitalWrite(gelbPin, LOW);
  delay(500);
 }
  for(int i=0; i<gruen; i=+1){
  digitalWrite(gruenPin, HIGH);
  delay(500);
  digitalWrite(gruenPin, LOW);
  delay(500);
 }
}
```

7. Bibliotheken, ADXL-Objekt, Variablen und Setup

   Das letzte, was das für ein funktionsfähiges Programm noch fehlt, sind sämtliche vorbereitenden Einstellungen bzgl. der Sensoren, die im Gesamtprogramm ergänzt sind.

8. Jetzt bist du dran: Mögliche Programmverbesserungen

   Das Programm ist bei weitem noch nicht optimal. Zum Beispiel lässt sich die Codeausgabe noch erheblich verbessern, globale Variablen können teilweise vermieden werden, ... Wenn du möchtest, kannst du sogar dem Würfel eine andere Aufgabenstellung vorgeben. Wie wäre es zum Beispiel hiermit: Wie beim Memory enthält der Würfel drei Seitenpaare. Wenn die spielende Person diese Pärchen in beliebiger Reihenfolge durch Drehen der entsprechenden Seiten nach oben verdeutlicht, hat sie die Aufgabenstellung erfüllt und erhält den Lösungscode.

   Viel Freude beim Experimentieren und Optimieren!