Eckart Modrow

Im Supermarkt mit SQLsnap

Abstract:

In diesem Artikel wird versucht, Erfahrungen mit potentiellen gesellschaftlichen Auswirkungen von Informatiksystemen direkt in den algorithmisch orientierten Fachunterricht zu integrieren. Die dafür erforderlichen Werkzeuge werden beschrieben und implementiert. Die Ergebnisse finden sich als SQLsnap unter *http://snapextensions.uni-goettingen.de*.

1. Die Ziele

Die Universität von Berkeley versucht in ihrem schönen Project „BJC: Beauty and Joy of Computing“ (BJC, 2013) Nicht-Informatik-Studierende so an das Programmieren heranzuführen, dass diese gesellschaftliche Folgen der Informatik besser beurteilen können als Studierende ohne diese Ausbildung. Grundlage dieses Projekts ist die Überzeugung, dass erst die Realisierung eigener Ideen mithilfe algorithmischer Methoden und geeigneter Programmierwerkzeuge, hier: Snap! (SNAP, 2014), die nachhaltige Basis schafft, sich dauerhaft und angemessen mit solchen Fragen zu beschäftigen, vor allem aber die „richtigen“ Fragen zu stellen. Die Beschäftigung mit den Grundlagen der Informatik soll es den Lernenden ermöglichen, nicht nur die aktuelle Technik zu beurteilen, sondern zukünftige Entwicklungen in diesem Bereich mit zu vollziehen. Zu diesem Zweck sind das bekannte Buch von Abelson, Ledeen und Lewis *Blown to Bits - Your Life, Liberty, and Happiness After the Digital Explosion* (BTB,2008) sowie umfangreiche Materialien zum Einsatz von Computern in verschiedenen Bereichen in den Kurs integriert. Programmieraufgaben wechseln mit Studien der entsprechenden Kapitel. Dabei sind die Aufgaben, soweit mit Anfängern möglich, eng an gesellschaftlich relevanten Fragestellungen wie der Funktion von Suchmaschinen oder Operationen auf Gensequenzen, orientiert. Anfangs sind die Beispiele sehr traditionell – wie in einem Anfängerkurs auch nicht anders zu erwarten. Bedingt durch die mächtigen Strukturen von Snap! geht es am Ende des Kurses aber weit über Standardbeispiele hinaus.

Die zahlreichen Beispiele aus *Blown to Bits* beruhen oft auf der Kombination unterschiedlicher Datenquellen, die früher strikt getrennt waren, heute aber im Internet bequem zugänglich sind. Beispiele sind der Zugriff auf Personendatenbanken zusammen mit Ergebnissen der Gesichtserkennung oder des Lesens von PKW-Kennzeichen. Nun macht sich gerade im Bereich der Bildverarbeitung die geringe Arbeitsgeschwindigkeit von Snap! sehr negativ bemerkbar. Schließlich ist das System in erster Linie mit der Kontrolle des Programmablaufs und der grafischen Oberfläche beschäftigt. Sind aber die Wartezeiten auf Arbeitsergebnisse zu groß, dann ist eine interaktive, experimentell orientierte Arbeitsweise, wie sie für selbstständige Schülerarbeiten typisch ist, kaum noch möglich. Ebenso fehlt der Zugriff auf Datenbanken oder Dateien in der Grundversion von Snap!. Glücklicherweise stellt das System aber Funktionalitäten bereit, mit deren Hilfe sich solche Zusätze realisieren lassen. Ziel der Arbeit ist es deshalb

- anhand eines Beispiels festzustellen, welche Funktionen zur Realisierung von Anwendungen, anhand derer sich gesellschaftliche Folgen der Informatik diskutieren lassen, noch fehlen
- und entsprechende Zusätze zu realisieren.

Wir stellen uns einen Supermarkt mit verschiedenen Abteilungen vor: einer Scannerkasse, die Barcodes auf den Produkten liest und Artikelnummern und Rechnungen liefert, einer Lagerverwaltung mit integrierter Datenbank, die Artikelnummern erhält und Preise ermittelt, sowie, falls nötig, Produkte von den Lieferanten bestellt, einer „intelligenten“ Waage, die mithilfe einer Kamera Früchte erkennt und Barcodes erzeugt, einer Werbeabteilung, die für Payback, Werbung, Sonderangebote, …

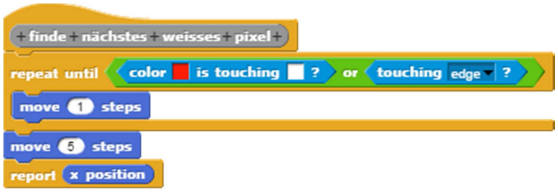
verantwortlich ist, und einer Sicherheitsabteilung, die für die Bezahlung der Parkgebühren sorgt und Kunden mit Hausverbot „betreut“. Weitere Abteilungen können natürlich dazu kommen. Die Implementationen der einzelnen Abteilungen laufen auf verschiedenen Computern und kommunizieren mithilfe von Textdateien auf einem Server. Und wir benutzen keine professionellen Verfahren, sondern nur „naive“ Lösungen, die zu Verbesserungen durch die Lernenden herausfordern.

Natürlich muss das Funktionieren der Abteilungen sichergestellt werden. Daraus ergeben sich dann Anforderungen an die Erweiterungen, zum Beispiel der Zugriff auf die oben genannten Textdateien. Die interessanten Fragestellungen ergeben sich allerdings erst aus der Kombination der Daten. Erfasst die Sicherheitsabteilung Daten der Besucher im Parkhaus, dann können diese zusammen mit den Daten der Kasse genutzt werden, um potentiell interessierten Kunden spezielle Angebote durch die Werbeabteilung zukommen zu lassen. Die vorhandenen Daten werden für den veränderten Zweck aber meist nur bedingt geeignet sein. Um die neuen Vorhaben besser zu realisieren, werden weitere Daten benötigt, die dann wieder zu neuen Nutzungsmöglichkeiten führen. Wir müssen deshalb nicht die Existenz von „Datenverbrechern“ annehmen, die zur Datensammelwut führt: es genügen ganz normale Menschen, die ihren Job optimal ausführen wollen. Deren Bedarf nach möglichst detaillierten Informationen kollidiert direkt mit den Persönlichkeitsrechten der Betroffenen – und daraus ergibt sich der Bedarf nach gesetzlichen Regelungen. Der politische Diskurs über diese Interessengegensätze und die daraus folgenden Regelungen können dann die Situation befrieden – so ist jedenfalls zu hoffen.

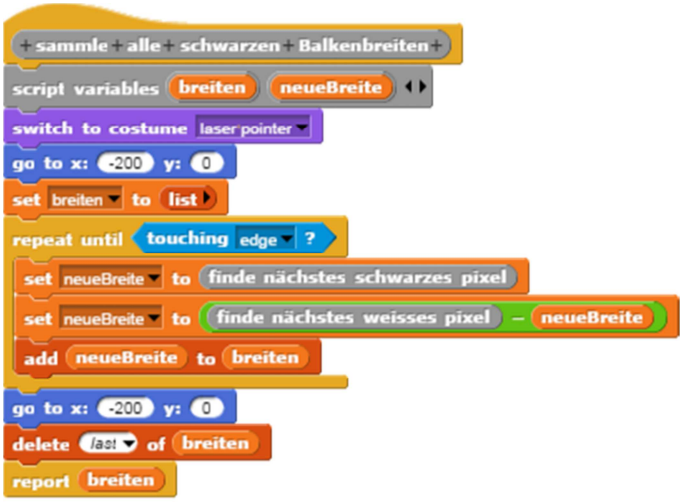
2. Die Scannerkasse

Wir beginnen mit einem Standardbeispiel, der Scannerkasse für Barcodes. Diese muss sich natürlich mit einem Datenbankserver verbinden können. Um diese und andere Möglichkeiten zum Datenbankzugriff müssen wir Snap! erweitern. Wir richten einen mySQL-Server ein und benutzen HTTP-Zugriffe. Dazu kann, wie in „Neues von BYOB / Snap!“ (MOD …) dargestellt, der http-Block innerhalb von Snap! benutzt werden oder man integriert entsprechende neue Blöcke.

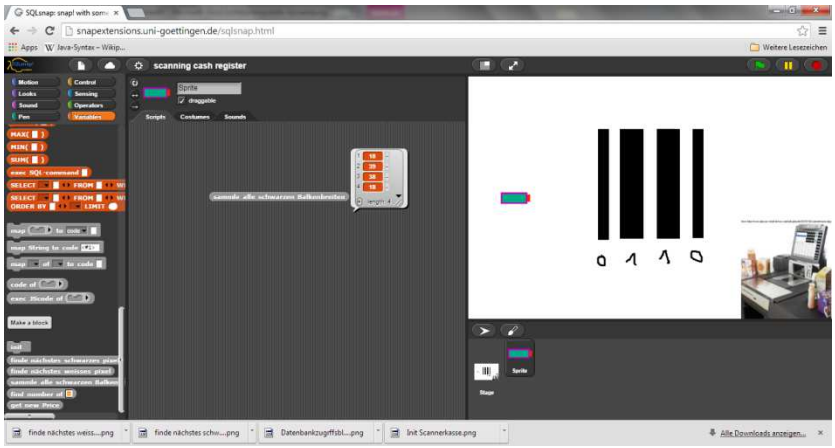
Wir wollen nicht gleich zu Anfang Barcodes als eigenes Thema behandeln und zeichnen deshalb nur einige schwarze Balken auf die Bühne, deren Breiten zu ermitteln sind. Dafür erhält die Turtle ein neues Kostüm, einen „Laserpointer“ mit einem roten Fleck vorne. Von dem kann man feststellen, ob er schwarze Balken berührt. Über die Positionen des Laserpointers lassen sich die Breiten der Balken ermitteln und in einer Liste ***breiten*** sammeln. Um das zu realisieren, schreiben wir zwei Methoden ***finde nächstes schwarzes pixel*** und ***finde nächstes weisses pixel***. Das ist im ersten Fall einfach, aber für die weißen Balken gibt es ein Problem. Wenn die Spitze der „roten Nase“ des Laserpointers einen schwarzen Balken berührt, hält er an. Aber der Rest der „Nase“ berührt immer noch den weißen Hintergrund. Wir müssen also den Laserpointer ein paar Schritte weiter bewegen, bevor wir nach weißen Pixeln fragen. Weiterhin soll der Suchprozess am rechten Bildrand stoppen.

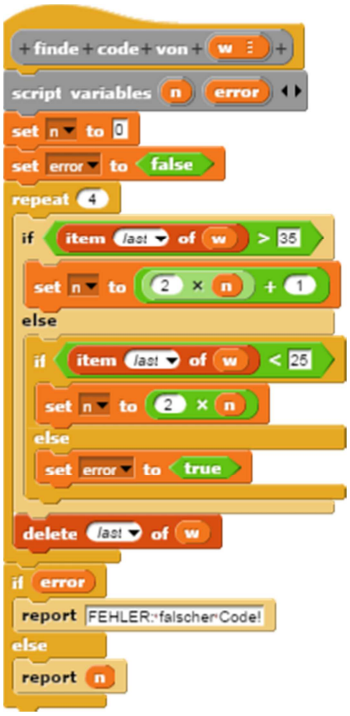


Nun ist einfach, die Breiten aller schwarzen Balken zu erhalten: Wir erzeugen zwei Variable (die Liste ***breiten*** und eine Zahl ***neueBreite***), messen die x-Positionen des linken und rechten Balkenrandes und packen die Differenz in die Liste. Wegen des Anhaltens am rechten Bildrand ist der letzte Messwert überflüssig und wird gestrichen.



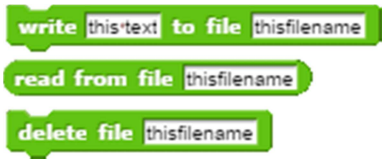
Wir benötigen einen Code, der anzeigt, was die Balken bedeuten. Ein Dualsystem mit vier Stellen ist dafür geeignet: ein breiter Balken (breite > 35) bedeutet „1“, ein schmaler (breite < 25) „0“.

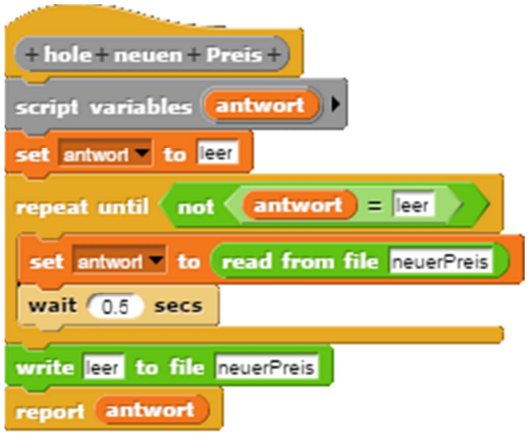




Um den numerischen Wert unseres Barcodes zu erhalten, wiederholen wir vier Mal: *Lies jeweils das letzte Listenelement und rechne es auf die übliche Art um. Dann lösche dieses Element. Falls es nicht im gewählten Bereich ist, melde einen Fehler.*

Wir senden den Wert des Barcodes an die Lagerverwaltung, um den aktuellen Preis zu erhalten, indem wir den ermittelten Code in eine vorher aus dem Wort „leer“ bestehenden Textdatei ***neuerCode*** auf dem Server schreiben und auf Antwort warten. Dafür nutzen wir die neuen Blöcke zur Bearbeitung von Textdateien auf dem Server:





Die Lagerverwaltung muss diese Datei regelmäßig lesen und bei Bedarf den richtigen Preis und die Bezeichnung des Artikels liefern, in einer Textdatei namens ***neuerPreis***. Danach setzt sie ***neuerCode*** wieder auf ***leer***. Die Scannerkasse liest nach der Änderung von ***neuerCode*** die Datei ***neuerPreis*** ebenfalls regelmäßig und erhält Preis und Bezeichnung, die zum Barcode gehören. Dann schreibt sie ***leer*** in diese Datei.

Damit haben die erste Abteilung rudimentär implementiert.

Natürlich gibt es Erweiterungsmöglichkeiten:

1. Echte Barcodes nutzen auch die Lücken zwischen den schwarzen Balken: als weiße Balken. Die Breite der weißen Balken hat die gleiche Bedeutung wie die der schwarzen. Die Methoden können entsprechend geändert werden.
2. Man kann ein Druckersprite entwickeln, das Barcodes auf die Bühne druckt. Zuerst muss der Benutzer natürlich nach der darzustellenden Zahl gefragt werden.
3. Man kann andere Barcodesysteme wählen und das Druckersprite entsprechend umbauen.
4. Die Kommunikation zwischen der Lagerverwaltung und der Kasse kann stark verbessert werden. Endlosschleifen sind nicht unbedingt die eleganteste Lösung.
5. Wenn die Lagerverwaltung richtig arbeitet, erhält die Kasse Antworten der Form <Preis>,<Bezeichnung>. Damit lassen sich Rechnungen produzieren, die Datum und Zeit sowie alle gekauften Produkte mit Preisen und die Gesamtsumme enthalten. Steuern sollen wie üblich angegeben werden.

Alle diese Möglichkeiten erweitern und vertiefen das Ausgangsthema, ohne den Kontext zu verlassen. Sie sind also ideal für differenzierende Gruppenarbeiten geeignet.

3. Die Lagerverwaltung

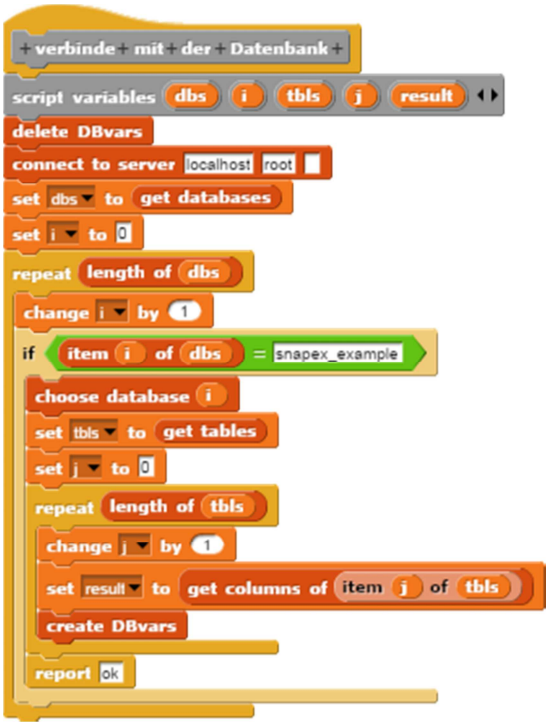
Unsere Lagerverwaltung benutzt eine MySQL-Datenbank auf dem Server. In diesem Fall die ***snapex_example*** Datenbank mit den Tabellen *products*, *distributors* und *prices*. Wir könnten beim Zugriff darauf die entsprechenden SQL-Anweisungen einfach als Text in den neuen Block ***execSQL-command***

eingeben – mit den damit verbundenen vielen Möglichkeiten für Tippfehler. Deshalb gibt es einige kleine Hilfen:

- die zwei Blöcke erzeugen für die Tabellen und die Spaltennamen der gerade ausgewählte Tabelle „Datenbankvariable“ der Form <Tabellenname> bzw. <Tabellenname>.<Spaltenname> oder löschen sie wieder. Diese besonderen Variablen erhalten als Wert ihren Namen, also die Aufschrift des Blocks, und können als Platzhalter interaktiv beim Erstellen von Abfragen benutzt werden.

- Die zwei Versionen des Select-Blocks dienen zur Erzeugung von einfachen bzw. fast vollständigen Select-Befehlen, die dann mit dem execSQL-command-Block ausgeführt werden können.

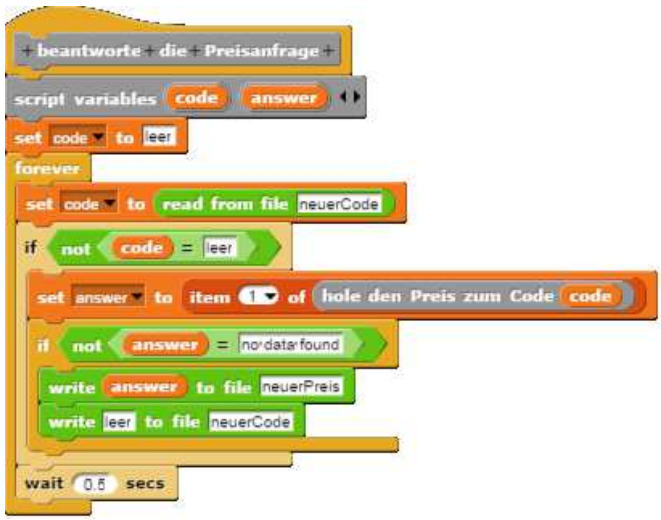
Die üblichen Operatoren und Aggregatfunktionen stehen ebenfalls als Blöcke zur Verfügung.



Ein typisches Skript, das die Verbindung mit der gesuchten Datenbank herstellt und dann die Datenbankvariablen erzeugt, sähe wie nebenstehend gezeigt aus.

Eine Anfrage, um den aktuellen Preis und die Produktbezeichnung zu finden, ist dann einfach zu bilden: wir erzeugen den Text der Anfrage, indem wir geeignete Variable im ***SELECT***-Block anordnen, und lassen ihn ausführen.

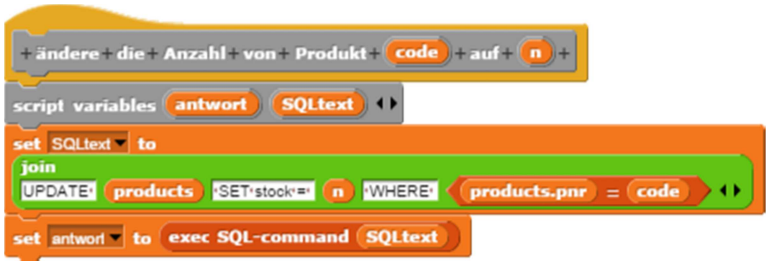
Die nicht terminierende Schleife ist im Block ***beantworte die Preisanfrage*** gekapselt. Wir sehen dort nach, ob sich die Datei ***neuerCode*** geändert hat. Falls das geschieht, fragen wir den SQL-Server nach dem Preis und der Bezeichnung und schreiben diese in die Datei ***neuerPreis*** - falls wir vom Server die richtige Antwort bekommen haben.



Wenn dieses Skript in der Lagerverwaltungs-Instanz von ***SQLsnap*** läuft, erhält die Scannerkasse die richtige Antwort.

Falls Sie Schreibrechte auf dem Server haben[1], können Sie die Lagerdaten auch verwalten. Zuerst müssen Sie die Anzahl der Produkte mit einem bestimmten Code ermitteln.

In SQLsnap haben wir derzeit nur Blöcke für ***select***-Kommandos. Deshalb benutzen wir den ***exec sql-command*** Block direkt für ein Update-Kommando.



Wenn der Code ein Produktschlüssel ist, können wir beide Blöcke zu einem Update-Kommando kombinieren:

Mit diesen Möglichkeiten lässt sich schon einiges anstellen:

1. Wenn einige Produkte verkauft worden sind, sinkt der Bestand unter den Mindestbestand. Neue Produkte müssen bestellt werden, sodass der Maximalbestand erreicht wird. Dafür sollte der Lieferant mit dem geringsten Preis gewählt werden.
2. Der Supermarkt möchte ein BIO-Supermarkt werden. Dafür müssen die Lieferanten entsprechend gewechselt werden, wenn das möglich ist, und die Preise sind anzupassen.
3. Bio-Produkte sollen zusätzlich zu den billigeren Produkten in die Datenbank eingeführt werden, sodass eine Auswahlmöglichkeit besteht.
4. Jeden Samstag soll ein Update-Durchlauf erfolgen. Wenn sich die Lieferantenpreise geändert haben, werden die Produktpreise entsprechend angepasst.
5. Der Supermarkt funktioniert, braucht aber Geld. Alle Preise werden um 10% erhöht.
6. Das Management benötigt Statistiken über die Verkäufe pro Monat und Jahr. Diese Daten müssen erhoben und in geeigneten Diagrammen dargestellt werden.
7. Es sollen Blöcke für die folgenden SQL-Anweisungen geschrieben werden:

   Syntax: UPDATE <tablename> SET column = value {,column = value} WHERE <condition>;
   Beispiel: UPDATE products SET stock = 99 WHERE pnr = 11;

   Syntax: INSERT INTO <tablename> (column{,column}) VALUES (value{,value});
   Beispiel: INSERT INTO prices (pnr,dnr,price) VALUES (1,2,3.45);

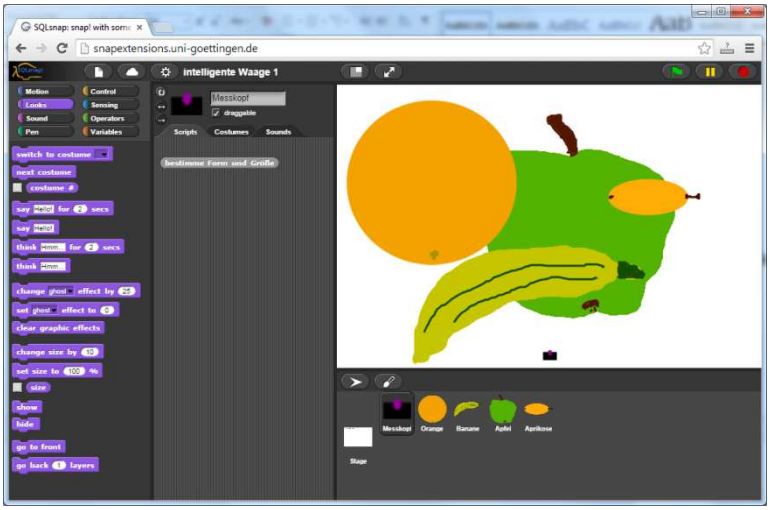
   Syntax: DELETE FROM <tablename> WHERE <condition>;
   Beispiel: DELETE FROM distributors WHERE name = 'Miller';

Isoliert betrachtet sind diese SQL-Möglichkeiten wesentlich unpraktischer als die Arbeit mit einem der üblichen SQL-Tools. Der Vorteil liegt darin, dass SQL-Anweisungen direkt erprobt und dann in Skripte eingebunden werden können, die Funktionalität für ein größeres Projekt bereitstellen. Es geht dann nicht darum, „SQL zu lernen", sondern mithilfe von SQL Probleme zu lösen.

[1] Falls nicht: installieren Sie mySQL auf dem Computer, auf dem die „Lagerverwaltungs-Instanz" von SQLsnap läuft. Verbinden Sie sich mit ***localhost*** als Benutzer ***root***. Sie haben dann Schreibrechte.

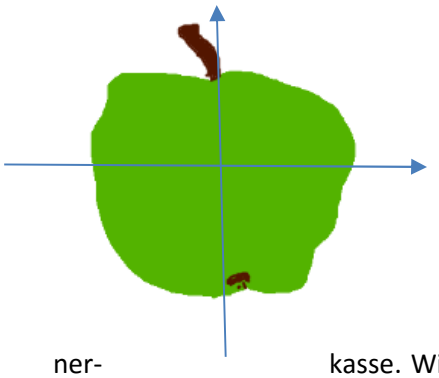
4. Eine „intelligente“ Waage

Das nächste Beispiel erfordert halbwegs schnelle Bildverarbeitung und lässt sich zu gesellschaftlich relevanten Fragestellungen etwa bei der Gesichtserkennung erweitern: einer Waage mit Kamera, die erkennt, welche Art von Früchten sich in der Waagschale befinden. Wir beginnen wieder möglichst einfach, indem wir verschiedene Früchte stark vereinfacht zeichnen und dann Kriterien entwickeln, anhand derer sich die Früchte unterscheiden lassen. Beginnen wir also mit einem Apfel, einer Orange, einer Aprikose und einer Banane.

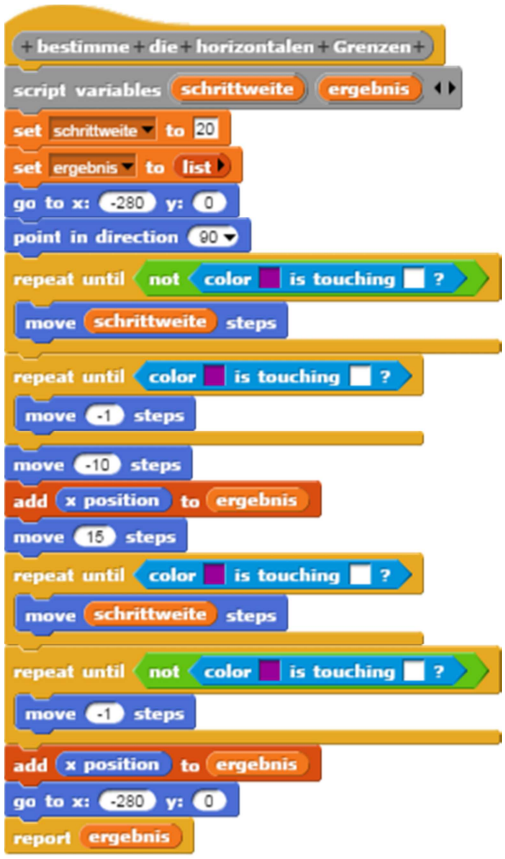


Die Unterschiede sind klar: Apfel und Orange sind rund, die Banane ist lang; Aprikose, Orange und Banane sind gelb-orange, unser Apfel ist grün; die Aprikose ist klein, die anderen sind größer. Aber was bedeutet „rund“, „lang“, „gelb“, grün“ oder „groß“? Wir wissen das, der Computer aber nicht. Wir müssen es ihm erst beibringen.

Die Unterscheidung von „groß“, „klein“ und „mittel“.

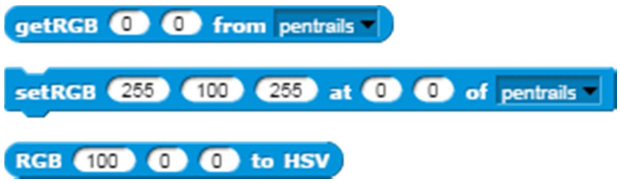


Wir bringen eine Frucht in die Mitte der Bühne und „drucken“ dort ihr Bild. Dann lassen wir einen Messkopf von links nach rechts und von unten nach oben laufen und bestimmen die Grenzen der Frucht. Das Verfahren kennen wir ja schon von der Scannerkasse. Wir berechnen das Verhältnis von Breite und Höhe, das bei „runden“ Früchten in der Nähe von 1 liegt und bei „langen“ sehr viel kleiner ist. Bei „ovalen“ Früchten sollten wir eigentlich in mehreren Richtungen messen, aber das überlassen wir den Perfektionisten. „Oval“ bedeutet bei uns „weder rund noch lang“.



Der relativ schnell arbeitende Block ***bestimme die horizontalen Grenzen*** liefert eine Liste mit zwei Werten: dem linken und rechten Rand der Frucht. ***Bestimme die vertikalen Grenzen*** funktioniert entsprechend. Mit den Ergebnissen von beiden können wir entscheiden, ob eine Frucht rund, oval oder lang ist, und wir haben auch deren Größe.

Schwieriger wird es mit den Farben der Früchte. Snap! arbeitet wie Scratch und BYOB mit einer Variante des HSV-Modells. Schlimmer noch: es gibt gar keine Möglichkeit, die Farbe eines Bildpunkts zu bestimmen. Also brauchen wir drei entsprechende neue Blöcke:



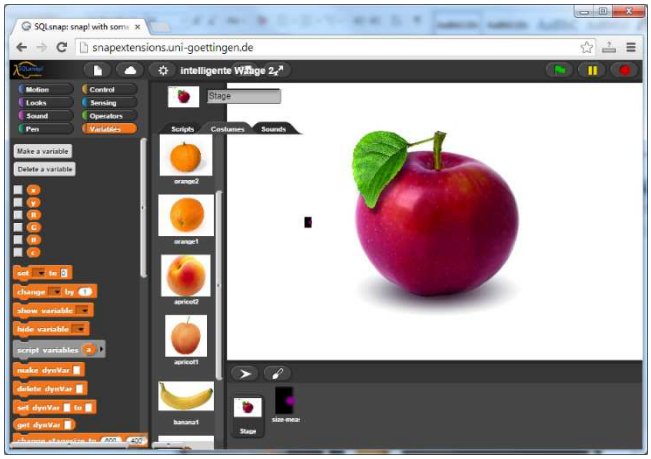
Bei den ersten beiden können wir auswählen, ob wir auf dem Kostüm des Bühnenhintergrunds RGB-Werte schreiben oder lesen wollen – oder ob wir dafür die gespeicherten Spuren des Zeichenstifts (*pentrails*) benutzen. Diese Alternative ermöglicht z. B. das Kopieren oder Ändern von Bildern, deren Original erhalten bleiben soll. In unserem Fall genügt das Lesen. Wir bestimmen die Farbe der Frucht an zehn unterschiedlichen Punkten auf den beiden Schnittlinien und bilden von diesen Messwerten den mittleren RGB-Wert. Diesen geben wir als Liste zurück.

Damit haben wir alle Werkzeuge zusammen, um die Eigenschaften einer Frucht zu bestimmen. Für den Apfel erhalten wir:

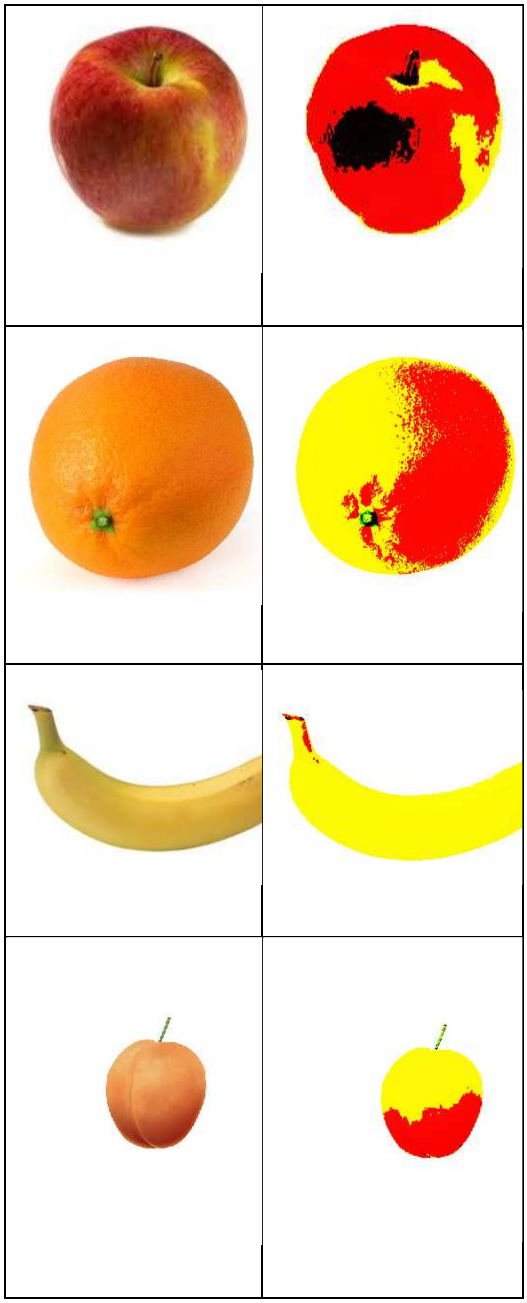
Legen wir diese Daten in der Datenbank ab, dann können wir die Preise der Früchte abrufen und Etiketten drucken.

Wie sieht es aber mit den Farben echter Objekte aus? Die sind ja meist nicht so schön einfarbig wie die gezeichneten. Wenn wir wie angegeben die mittlere Farbe einer Orange bestimmen, dann erhalten wir als Ergebnis einen von 16.777.218 möglichen Werten. Das sind etwas viele. Wir müssen die Anzahl der möglichen Farben reduzieren.

Versuchen wir es mal so: Für jeden der drei RGB-Werteentscheiden wir, ob er „groß“ oder „klein“ ist. Im ersten Fall ändern wir ihn auf 255, im zweiten auf 0. Damit erhalten wir 8 mögliche Farben. Damit probieren wir aus, ob wir noch genügend relevante Details auf Früchtebildern erhalten. Um genügend Platz für solche Fotos zu haben, ändern wir die Bühnengröße mit dem neuen Block **change stagesize** auf 400 x 400 Pixel.



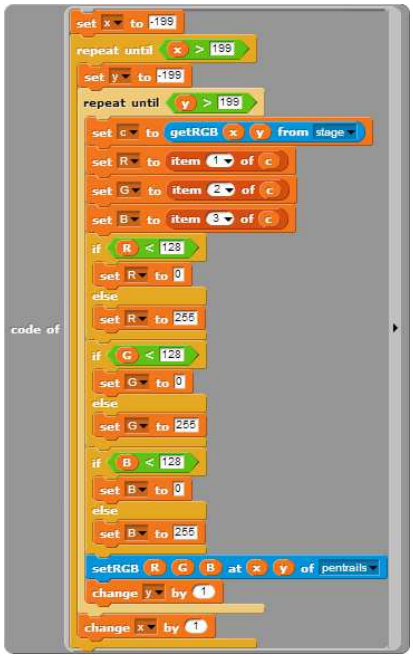
Danach wählen wir einige Früchtebilder als Bühnenhintergründe und reduzieren den Farbraum wie unten beschrieben.



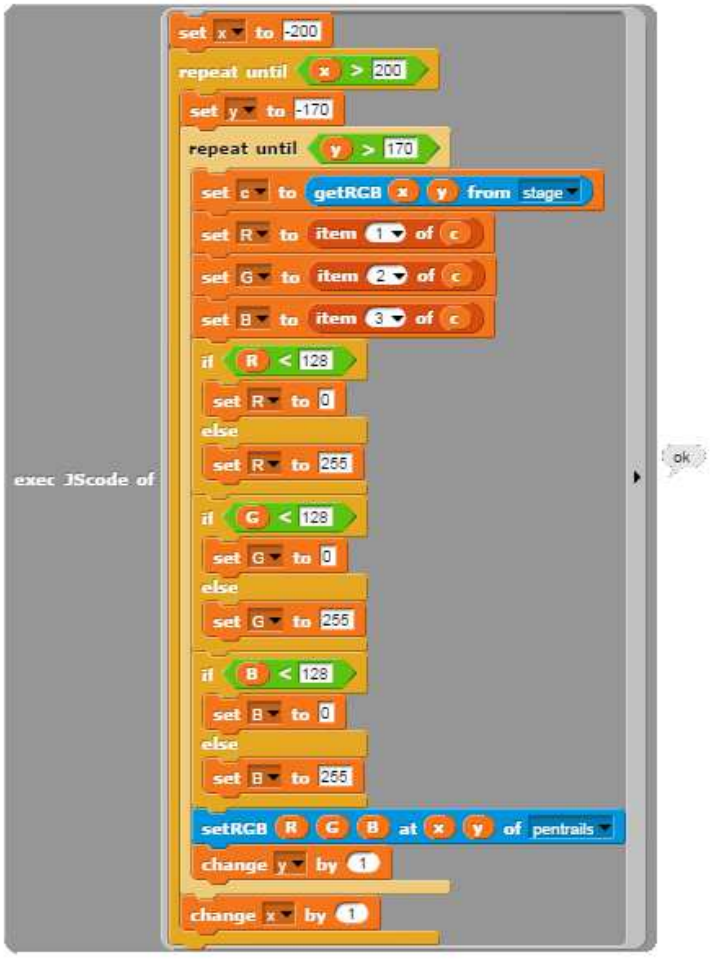
Die Ergebnisse sind gar nicht so schlecht. Aber wie erhält man sie?

Wenn wir ein Skript ***Farbraumreduzierung*** laufen lassen, dann funktioniert es schon, aber es läuft unglaublich lange. Deshalb nutzen wir die Fähigkeit von Snap!, den einzelnen Blöcken Code zuzuordnen. Wählen wir dafür JavaScript, dann steht im Browser ein entsprechender Interpreter zur Verfügung, dem wir den mit dem ***code of*** – Block erzeugten Code „durchreichen“ können.

```
x = -199;
do{
   y = -199;
   do{
      c = getPoint(x,y,"stage");
      R = (c.length() < 1 ? new String() : (String("1")==="any") ? c.at(Math.floor(Math.random() * (c.length()) + 1)) :
                                           (String("1")==="last") ? c.at(c.length()) :
                                           1 < 1 ? new String() :  c.at(1));
      G = (c.length() < 1 ? new String() : (String("...
```

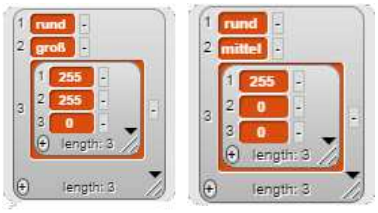


Die Ausführung des erzeugten Codes erfolgt durch den neuen ***exec JScode*** – Block. Wenn wir das Skript dort ausführen, erhalten wir das geänderte Bild innerhalb von Sekunden. Dabei sollten wir beachten, dass wir das hohe Tempo mit Verlusten an Sicherheit erkaufen: eventuelle Endlosschleifen z. B. können nicht mehr unterbrochen werden, weil die Kontrolle der Programmausführung nicht mehr bei Snap! liegt. Wir sollten dem schnellen Block deshalb nur gut getestete Skripte übergeben. und vorher das Programm sichern.



Den ***execJScode*** - Block integrieren wir in den Block Farbraumreduzierung und gewinnen einen sehr schnellen Block zur Bildverarbeitung.

Die mittleren Farben liegen natürlich meist außerhalb unseres Mini-Farbraums von acht Werten. Wir schreiben deshalb einen Reporter-Block zur Reduktion eines einzelnen Farbwerts und wenden den auf die ermittelten Farbwerte an. Damit erhalten wir für eine Orange und für einen kleineren roten Apfel die nebenstehenden Werte.



Interpretieren wir den Farbwert 255 als „1“ und den Farbwert 0 als „0“, dann können wir unseren acht Farben einfache Farbcodes zuweisen, die sich leicht in einer Datenbank ablegen lassen: Rot hätte den Code 100, also 4. Gelb den Code 6 und Blau den Code 1.

Insgesamt haben wir jetzt eine Werkzeugkiste zur Früchteerkennung zusammen, aus der sich das folgende Vorgehen ergibt:

1. Wählen Sie das Foto einer Frucht auf weißem Hintergrund als Bühnenkostüm. Solche Fotos lassen sich leicht mit dem Smartphone oder der Laptopkamera anfertigen.
2. Reduzieren Sie den Farbraum des Bildes.
3. Messen Sie die Form und die Größe der Frucht.
4. Messen Sie die mittlere Farbe der Frucht und reduzieren Sie diesen Farbwert nochmals.
5. Bestimmen Sie den Farbcode der Frucht.
6. Legen Sie die gewonnenen Daten in einer Datenbank ab oder bestimmen Sie die Art der Frucht aus einer Datenbankabfrage.

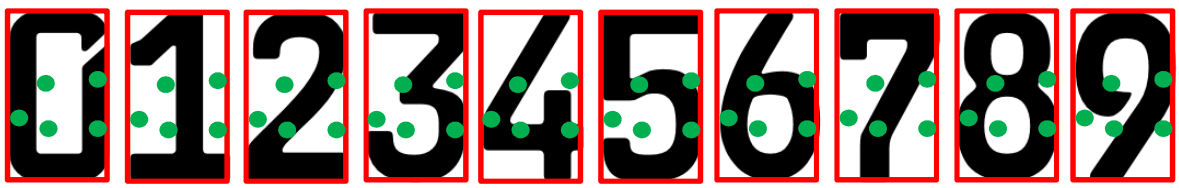
Mit unseren sehr einfachen Werten lassen sich dann 3 * 3 * 8 = 72 Früchte unterscheiden. Das sollte für eine normale Fruchtabteilung genügen.

4. KFZ-Kennzeichenerkennung in der Sicherheitsabteilung

Bisher kamen gesellschaftlich relevante Aspekte eigentlich nicht vor. Wir ergänzen deshalb den Supermarkt um eine engagierte Sicherheitsabteilung, die u.a. für den Betrieb des Parkhauses verantwortlich ist. Um die Abrechnung dort zu vereinfachen, sollen die Kennzeichen der einfahrenden Autos automatisch erkannt werden. Registrierte Kunden brauchen dann nicht an der Eingangsschranke zu warten, sondern können einfach durchfahren.

KFZ-Kennzeichen benutzen einen speziellen Zeichensatz, der gut zur automatischen Zeichenerkennung geeignet ist – und sie haben einen schwarzen Rand. Mit dessen Hilfe lassen sich Nummernschilder auf Bildern finden – und auf diesen dann die Zeichen. Damit haben wir innerhalb des Gesamtproblems eine Reihe von Teilproblemen gefunden, die sich für unabhängig arbeitende Schülergruppen eignen, unterschiedlich anspruchsvoll sind und zusammen ein Problem lösen, dessen Auswirkungen evident sind. Vor allem aber: die Probleme lassen sich – mit gutem Willen – auf sehr elementarem Niveau lösen. Wir benötigen keine ausgefeilten Verfahren. Im Gegenteil: viele Teillösungen können wir von der Waage übernehmen. Wir können also Nummernschilder auf Bildern suchen lassen – mit und ohne Berücksichtigung der Perspektive –, Bilder verschmutzter Kennzeichen aufbereiten, Länderkennungen (die blauen Bereiche) analysieren, und natürlich die Zeichen erkennen. Wir wollen uns hier auf den letzten Punkt, und dort wiederum auf die Ziffern, beschränken, und dafür ein möglichst einfaches Verfahren wählen, das nach Verbesserungen durch die Lernenden geradezu schreit.

Weil alle Nummernschildzeichen die gleiche Größe haben, lassen sie sich Zeichen leicht finden, wenn wir das Kennzeichen mit seinem schwarzen Rand erst einmal gefunden haben. Wir suchen dann innerhalb des Randes von links nach rechts nach senkrechten Linien, auf denen sich schwarze Pixel befinden, und danach solche, die ganz weiß sind. Zwischen diesen findet sich ein Zeichen. Entsprechend lassen sich die vertikalen Grenzen ermitteln. Mit den erhaltenen Werten lässt sich ein Fenster angeben, das ein Zeichen umschließt. Die Verfahren dazu kennen wir von der Waage, neu ist nur das OCR-Problem, Zeichen innerhalb des Fensters zu unterscheiden. Dazu lassen sich unterschiedliche Verfahren erfinden, z. B. durch Vergleich der weißen und schwarzen Pixelzahlen im gesamten Fenster oder in Teilen davon. Die Treffsicherheit der Verfahren lässt sich experimentell ermitteln. In unserem Fall wollen wir eine Art „Sensorfeld“ benutzen, das an unterschiedlichen Fensterpositionen misst, ob dort weiße oder schwarze Pixel zu finden sind. Diese „Sensoren“ lassen sich ganz gut so platzieren, dass man mit wenigen (hier: 5) die Ziffern sicher unterscheiden kann. Interpretieren wir die Messwerte wieder als Dualzahlen, dann lassen sich die Zeichencodes wie bei der Waage in einer Datenbank unterbringen.



Nebenbei haben wir ein System gefunden, das sich sehr gut dafür eignet, Zeichen zu „lernen“, und sich so z. B. an unterschiedliche Zeichensysteme selbstständig anpassen kann.

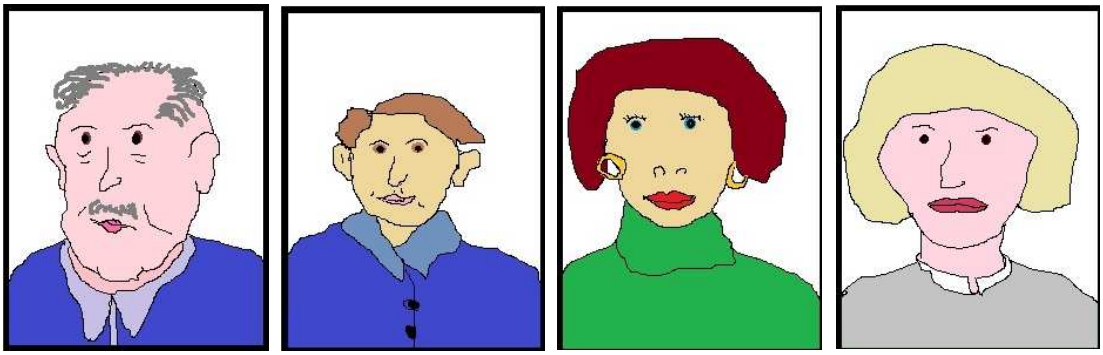
Wir wollen uns hier nicht mit den erforderlichen Skripten abgeben – die bringen wenig Neues -, sondern prüfen, was sich mit den ermittelten Daten anstellen lässt.

1. Die Werbeabteilung möchte häufige Kunden als VIP-Kunden auszeichnen. Diese dürfen in speziellen Bereichen nahe dem Fahrstuhl parken. Schreiben Sie eine SQL-Anfrage, die VIP-Kunden ausgibt.
2. Nach einiger Zeit ist der VIP-Bereich weitgehend mit Autos von Rentnern und Arbeitslosen belegt. Deshalb werden die Kaufumsätze der Kunden zu den VIP-Kriterien hinzugefügt. Da alle mit Kreditkarten zahlen, ist das kein Problem. Ändern Sie die SQL-Abfrage entsprechend.
3. Die Werbeabteilung möchte nun nicht nur wissen, welchen Umsatz ein Kunde bringt, sondern auch, was er gekauft hat. Mit diesen Informationen kann sie ihn mit spezielle Werbeaktionen und Sonderpreisen erreichen. Ändern Sie die Struktur der Datenbank so, dass diese Aufgaben zu erfüllen sind.
4. Die Werbeabteilung möchte wissen, ob ihre Aktionen erfolgreich sind. Erreicht sie die Kunden? Versuchen Sie diese Antwort, basierend auf den verfügbaren Daten, zu beantworten.
5. Auf Autobahnen soll die LKW-Maut über Barrieren ermittelt werden, auf denen sich Geräte zum Lesen der Nummernschilder befinden. Diese lesen alle Kennzeichen und löschen die von PKWs. Die Ergebnisse dürfen nur zur Mautberechnung herangezogen werden. Ist das angemessen? Diskutieren Sie die Konsequenzen, falls alle Kennzeichen, Zeiten und Positionen gespeichert würden.

Wie man sieht, kommt man recht zwanglos von unverdächtigen Aufgabenstellungen zu ziemlich brisanten Konstellationen, die z.B. aktuell anlässlich von Verbrechen auf den Autobahnen in der Presse diskutiert wurden.

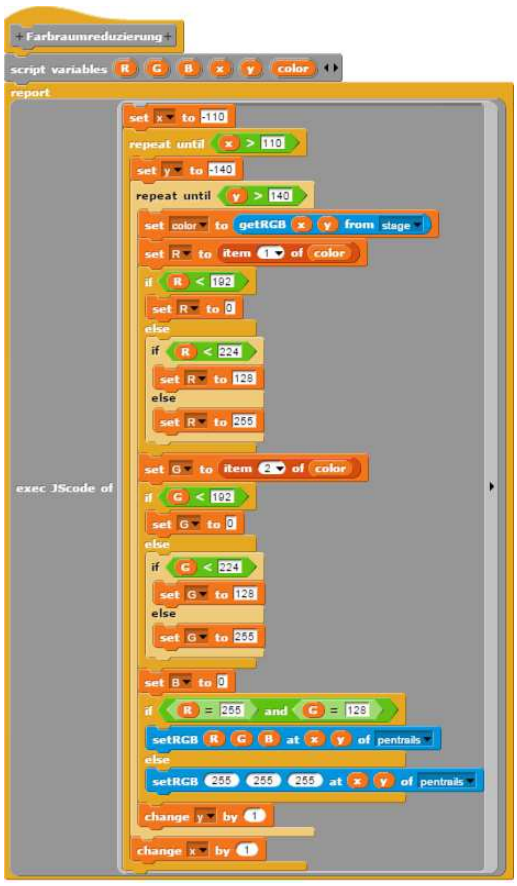
5. Die Werbeabteilung

Unsere Werbeabteilung ist begeistert von den Möglichkeiten der Nummernschilderkennung und möchte diesen Bereich ausweiten. Sie will wissen, wer sich alles im Supermarkt befindet. Dazu sollen die Kunden mit einem Programm zur Gesichtserkennung identifiziert werden. Versuchen wir das doch einmal so ähnlich wie bei der Früchteerkennung. Wir beschränken uns dabei auf ein paar gezeichnete Gesichter, die ähnlich wie auf Passfotos positioniert sind. Für diese ist ja aus guten Gründen eine bestimmte Haltung vorgeschrieben.



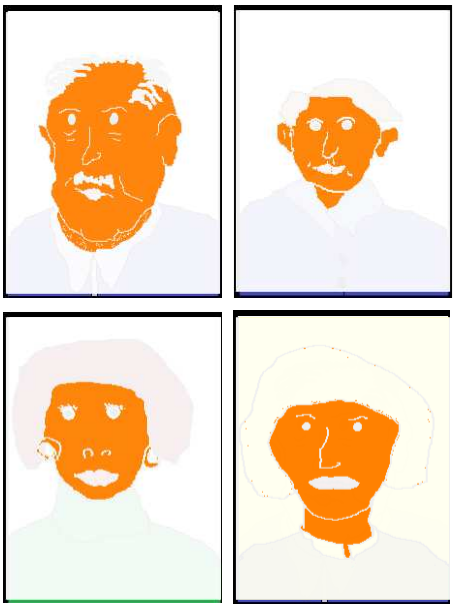
Paul Peter Mary Hannah

Um die Gesichter auf den Bildern zu finden, führen wir eine leicht modifizierte Farbraumreduzierung durch: wir ignorieren den Blauwert und reduzieren die Rot- und Grünkanal auf drei mögliche Werte, 0, 128 und 255.



Danach löschen wir alles bis auf Orange, der Rest wird weiß.

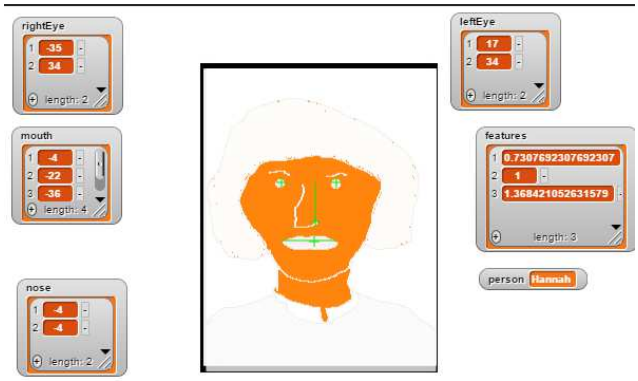
Wir erhalten:

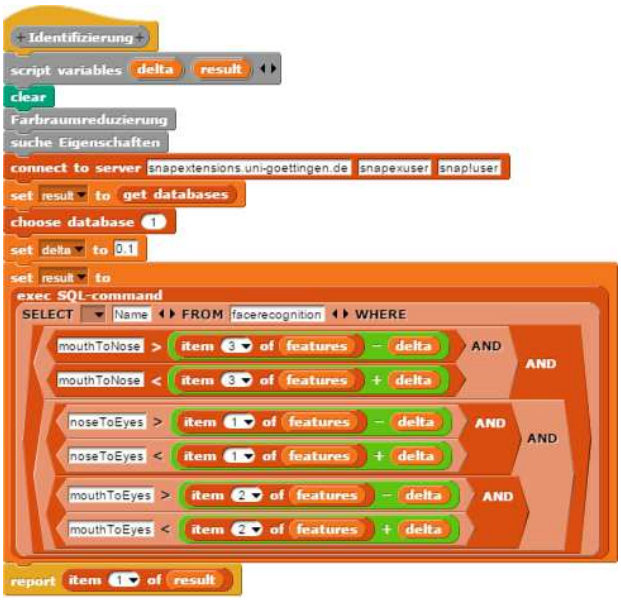


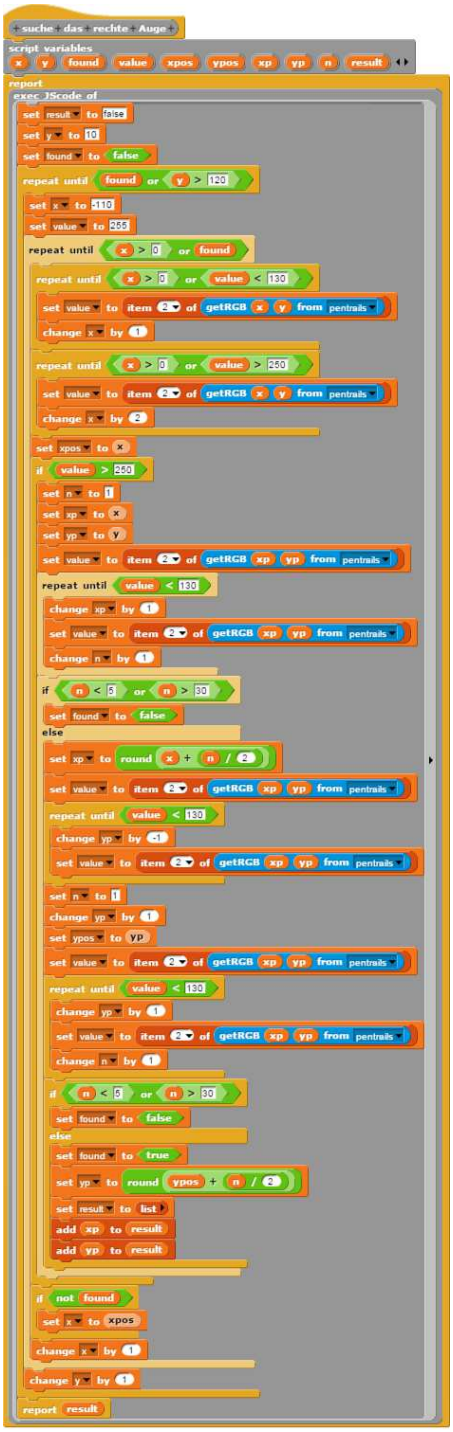
Nachdem wir die Gesichter gefunden haben, brauchen wir Parameter, um sie zu unterscheiden – genauso wie bei den Früchten. Dazu können wir z.B. den Augenabstand messen und ihn mit dem Abstand zum Mund vergleichen, oder mit der Nasenlänge. Der Fantasie sind keine Grenzen gesetzt. Bei Augen und Mündern handelt es sich um Löcher im orangenen Bereich. Das rechte Auge sollte sich oberhalb und links von der Mitte befinden, schließlich handelt es sich um Passfotos! Haben wir das gefunden, ist das linke nicht mehr weit. Und der Mund sollte sich in der Mitte darunter befinden. Die Skripte dafür erfordern einiges an Rechenaufwand und werden deshalb wieder dem JavaScript-Block übergeben.

Als Beispielskript ist hier ***suche das rechte Auge*** angegeben, das exzessiv von Zugriffen auf die Pixelwerte Gebrauch macht.

Wir messen die angegebenen Werte und zeichnen dabei gleich Markierungen in die Bilder, um die Ergebnisse bewerten zu können. Sie sind erstaunlich gut. Die Ergebnisse werden danach mit den Einträgen einer Datenbank verglichen, aus der sich der zugehörige Name ergibt.







Wiederum können wir unser Beispiel ausbauen. Wenn der Übergang von gezeichneten Früchten zu Fruchtfotos so einfach war, weshalb sollte es dann nicht mit echten Passfotos gelingen?

1. Die vier bisher benutzten Bilder sind sehr einfach. Experimentieren Sie mit echten Bildern. Präparieren Sie diese so, dass sich die Skripte darauf anwenden lassen.
2. Suchen Sie zusätzliche Parameter, um Gesichter zu unterscheiden.

Aber wir können natürlich auch „zickig“ werden, und die gewonnenen Daten anders benutzen.

3. Die Sicherheitsabteilung soll “unerwünschte Personen, also Diebe, Landfahrer, …vom Supermarkt fern halten. Wenn die Gesichtserkennung solche Personen, deren Daten natürlich in einer Datenbank gespeichert sein müssen, erkennt, dann lost sie einen Alarm aus. Manchmal produziert das Verfahren lautstarken Ärger, deshalb möchte die Sicherheitsabteilung die Personengruppe etwas subtiler fernhalten: die Garagenschranke öffnet sich für sie nicht, der Fahrstuhl streikt, Türen bleiben geschlossen, … Diskutieren Sie diese Situation.
4. Die Werbeabteilung hat auch nette Ideen. Es gibt zahlreiche Leute, die sich im Supermarkt aufhalten, aber nichts oder nur wenig kaufen. Andere kaufen nur Sonderangebote oder Billigprodukte. Diese werden ebenfalls zu „unerwünschten Personen“ deklariert, weil sie Platz beanspruchen, der besser für VIP-Kunden frei gehalten wird. Diskutieren Sie diese Situation.

Und es kann auch richtig gefährlich werden.

5. Unerwünschte Personen müssen ja erst einmal auffallen, bevor man sie schikanieren kann. Deshalb stellen Sicherheits- und Werbeabteilung zusammen Profile zusammen, anhand derer man sie identifizieren kann, bevor sie den Supermarkt das erste Mal betreten. Entwickeln Sie solche Profile und diskutieren Sie die Konsequenzen.
6. Die Werbeabteilung weiß durch die Kasse, was Kunden kaufen. Viele Kunden interessieren sich aber deutlich für Produkte, ohne sie zu kaufen. Deshalb soll der Weg der Kunden durch den Supermarkt verfolgt werden. Bleiben sie irgendwo besonders lange stehen, kann das einen unerfüllten Kaufwunsch signalisieren. Personalisierte Werbung für die entsprechenden Produkte kann den Kunden auf ihr Smartphone gesendet werden, oder die Daten dieser Kunden können an Geschäfte verkauft werden, die auf diese Produkte spezialisiert sind. Diskutieren Sie diese Situation.
7. Der Supermarkt will sich auf die VIP-Kunden konzentrieren. Diese werden wiederum über entsprechende Profile identifiziert (Automarke, Wohnviertel, persönliche Kriterien, die aus der Gesichtserkennung abgeleitet werden, Kaufverhalten, …). Um Ärger zu vermeiden, sollen Nicht-VIP-Kunden weiterhin in den Supermarkt gelassen werden, aber diese erfahren kleine Schikanen (s.o.). Diskutieren Sie diese Situation.
8. Gesichtserkennung ist immer dann möglich, wenn eine Kamera vorhanden ist, also in Smartphones, „smart glasses“, Laptops, Überwachungskameras, … Weil auch das Internet fast überall verfügbar ist, können die Bilder mit denen in zugänglichen Sozialen Netzwerken, Datenbanken, … verglichen werden; zugänglich für den Fotografen oder zugänglich für andere, die an die Bilder kommen. Deshalb kann in absehbarer Zeit jeder identifiziert werden, der ins Gesichtsfeld einer Kamera kommt. Diskutieren Sie diese Situation aus unterschiedlichen Sichten.

6. Fazit

Snap ist, obwohl noch nicht fertig, außerordentlich geeignet, den algorithmisch orientierten Teil der Schulinformatik abzudecken. Für die üblichen Anwendungen ist es inzwischen schnell genug. Durch die Möglichkeit des Zugriffs auf externe Server kann es um Funktionalitäten erweitert werden, die eigentlich außerhalb einer Browseranwendung liegen. Das wird inzwischen meist anhand der Robotersteuerung demonstriert, aber wie gezeigt sind auch Datenbankzugriffe leicht zu realisieren. Sogar die Grafikanwendungen können sehr deutlich beschleunigt werden, wenn die Kontrolle an JavaScript übergeben wird. Inzwischen sind einige der hier realisierten Features schon in Snap! integriert. Da somit fast alle Bereiche der Schulinformatik mit diesem Werkzeug ohne spezielle Syntaxkenntnisse bearbeitet werden können, kann die gewonnene Zeit gut für Projektarbeit genutzt werden. Die Informatik gewänne damit einen Teil ihres Charmes aus der Zeit vor dem Zentralabitur zurück, das ja individuelle Profilierungen nicht gerade fördert. Vor allem aber können die Projekte so gewählt werden, dass sie den Zugang zu aktuellen Fragestellungen liefern. Die handlungsorientierte Auseinandersetzung mit diesen sollte die Lernenden befähigen, nicht nur die aktuelle Situation, sondern auch zukünftige Entwicklungen kritisch zu hinterfragen, weil die Kritikfähigkeit aus Grundlagenkenntnissen erwächst. BJC leistet auf diesem Weg wegweisende Arbeiten, die sich mit den genannten Erweiterungen leicht auf das deutsche Schulsystem übertragen lassen. Ich würde mich freuen, auf diesem Weg MitstreiterInnen dafür zu gewinnen.

BJC 2013 http://bjc.berkeley.edu/

BTB 2008 Blown to Bits: Abelson, H., Ledeen, K., Lewis,H., http://www.bitsbook.com/

SNAP 2014 http://snap.berkeley.edu/

MOD … Neues in Snap!