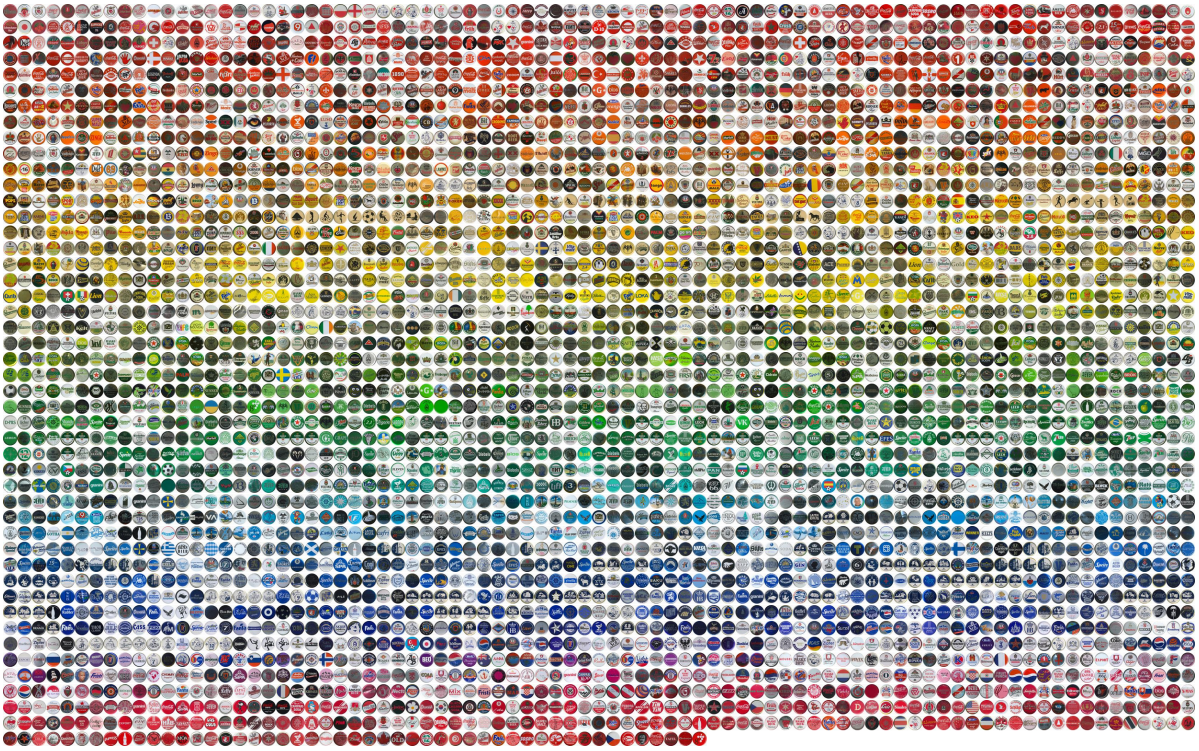

Kronis

Kronkorkenmanagement und Dublettensuche

Jörg J. Buchholz

10. Mai 2018

Teil I

Bedienungsanleitung

1 Einführung

Kronis [1] erfasst Kronkorken, sucht nach Dubletten in einer Datenbank und stellt die Kronkorken und ihre Daten im Netz dar.

Sammler stehen bei mehreren tausend[1] Sammelobjekten häufig vor dem Problem, ihre Schätze übersichtlich zu katalogisieren, zu verwalten und anderen Sammlern angemessen zu präsentieren. Bei umfangreichen Sammlungen verschwinden viele Kronkorken unbeachtet in Schränken, Kisten, Kartons, Tüten, Ordnern oder Sammelmappen. Und selbst wenn sie so dekorativ in Rahmen ausgestellt werden wie in Abbildung 1.1, bleibt die Zahl der Betrachter und der potenziellen Tauschpartner naturgemäß doch recht begrenzt. Aus diesem Grund sind fast alle Sammler mittlerweile dazu übergegangen, ihre Kronkorken zu fotografieren oder einzuscannen und mehr oder weniger geordnet ins Netz zu stellen.

Das nächste für den Sammler zu lösende Problem besteht dann in der Frage: „Hab ich den schon?", wann immer er eines neuen Objektes seiner Begierde habhaft wird. Nicht nur muss er sich daran erinnern, ob er den Bockbier-Kronkorken seiner Lieblingsbrauerei denn schon besitzt; er muss auch noch ganz genau wissen, wie dieser aussieht. Brauereien verändern im Laufe der Jahre das Aussehen ihrer Kronkorken manchmal nur geringfügig (leicht andere Schriftart, „moderneres" Logo, ...). Für einen Sammler ist das dann aber ein komplett „neues" Sammelstück. Die Lösung heißt halbautomatische Dublettenerkennung, bei der ein Programm diejenigen Kronkorken aus der Datenbank heraussucht, die dem Kandidaten am ähnlichsten sind und der Sammler durch sorgfältigen Vergleich entscheidet, ob der fragliche Kronkorken schon dabei ist.

[1] Der Rekord liegt momentan bei über 200 000 verschiedenen Kronkorken aus über 200 Ländern. [2] Im Vergleich dazu besteht das (hoch aufgelöste) Titelbild dieser Arbeit gerade einmal aus 3 867 Kronkorken.

Abbildung 1.1: Wand mit Kronkorkenbildern

2 Übersichtsseite

Kronis besteht aus zwei Webschnittstellen. Die erste ist die in Abbildung 2.1 dargestellte Übersichtsseite, die zweite wird in Kapitel 4 vorgestellt und ermöglicht die Erfassung der Kronkorken.

Abbildung 2.1: Übersichtsseite von Kronis

Auf der Übersichtsseite [1] können wir eine beliebige Auswahl der in der Datenbank

vorhandenen Kronkorken, nach verschiedenen Kriterien sortiert, als Bild und/oder Text, übersichtlich darstellen lassen. Beim Aufruf der Seite ist als Getränk Bier Alkoholfrei eingestellt, um die Anzahl der vom Server zu übertragenden Kronkorkenbilder klein zu halten.[1]

Als Sortierkriterium ist die mittlere Farbe des Kronkorken voreingestellt und es werden nur Bilder der Größe 40×40 Bildpunkte dargestellt. Wenn wir in der Drink-Auswahlliste statt Bier Alkoholfrei den leeren Eintrag (ganz oben in der Liste) auswählen und die Schaltfläche Display new selection betätigen,[2] werden alle in der Datenbank vorhandenen Kronkorken ausgegeben und wir sehen das Titelbild dieser Arbeit.

2.1 Tooltip und Details

Wenn wir mit dem Mauszeiger einen kleinen Augenblick über einem Kronkorken verweilen, erscheint ein kleines Fenster, in dem die Daten dieses Kronkorkens angezeigt werden. In der in Abbildung 2.2 dargestellten Bildschirmkopie ist der Mauszeiger (über dem fünften Kronkorken) nicht zu sehen.

Abbildung 2.2: Tooltip

Durch das Anklicken eines einzelnen Kronkorkens auf der Übersichtsseite (Abbildung 2.1) gelangen wir auf seine in Kapitel 3 beschriebene Detailansicht.

2.2 Auswahlkriterien

Die Menge der dargestellten Kronkorken können wir beliebig beschränken und filtern, indem wir verschiedene Auswahlkriterien (Selection criteria) – auch in Kombination mit-

[1] Das Übertragen und Rendern tausender Bilder und Bildinformationen kann auch bei schnellen Datenleitungen schon mal ein Minütchen dauern. Interessanterweise ist dabei momentan Firefox um ein Vielfaches schneller als Chrome.

[2] Wir müssen die Schaltfläche Display new selection jedes Mal neu betätigen, nachdem wir Auswahl- oder Sortierkriterien geändert haben, um die neue Liste anzuzeigen.

einander – verwenden. Jede Auswahlliste – wie beispielsweise die in Abbildung 2.3 dargestellte Getränkeauswahlliste – kann dabei viele hundert Einträge umfassen.[3]

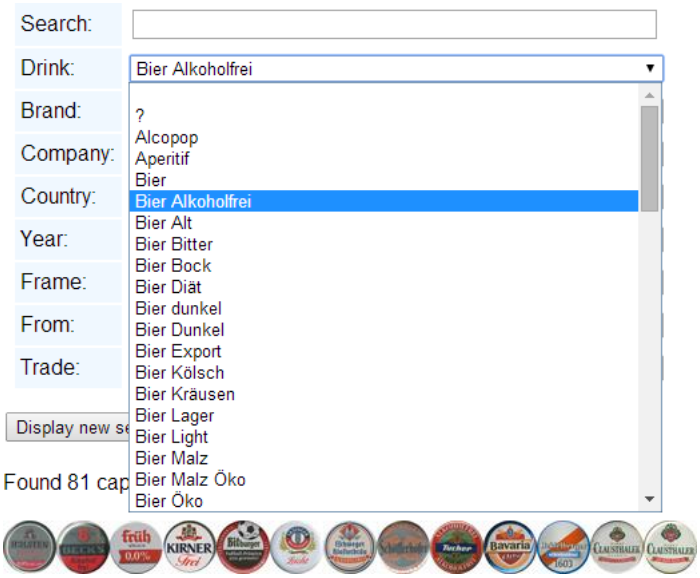

Abbildung 2.3: Auswahlliste Getränk

Der erste Eintrag jeder Liste ist leer. Wir wählen den leeren Eintrag, wenn die jeweilige Liste die Auswahl der Kronkorken nicht einschränken soll. Das Fragezeichen im zweiten Eintrag zeigt uns alle Kronkorken, für die das jeweilige Kriterium unbekannt ist; in der Getränkeliste sind das also alle Kronkorken, von denen wir nicht wissen, auf welchem Getränk der Kronkorken mal gesessen hat.

Alle Auswahlkriterien werden mit einem logischen UND verknüpft. Dies bedeutet, dass wir nur die Kronkorken sehen, die gleichzeitig alle ausgewählten Kriterien erfüllen. Wenn wir also sowohl das Getränk Bier Alkoholfrei als auch den Hersteller Warsteiner Brauerei auswählen, werden statt aller alkoholfreien Kronkorken (Abbildung 2.1) nur die vier alkoholfreien Kronkorken von Warsteiner dargestellt (Abbildung 2.4).

[3]Bei der Erfassung (Kapitel 4) eines neuen Kronkorkens werden uns die schon vorhandenen Einträge eines Auswahlkriteriums angeboten, um Doppeleinträge zu verhindern. Wir können aber natürlich auch selbst einen ganz neuen Eintrag erzeugen.

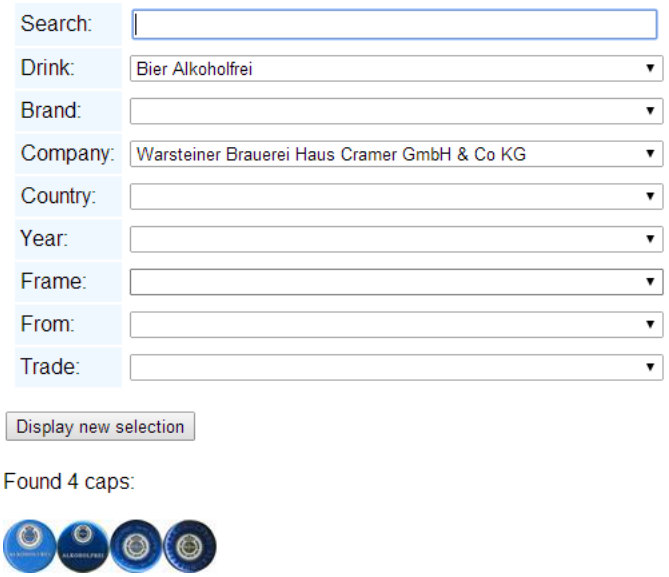

Abbildung 2.4: Nur die vier alkoholfreien Warsteiner-Kronkorken herausfiltern

Eine Besonderheit stellt das letzte Auswahlkriterium (Trade) dar. Dort können wir nur entscheiden, ob wir die Auswahl auf diejenigen Kronkorken einschränken wollen, die für einen Tausch zur Verfügung stehen (Trade caps only).

Wenn wir nicht so ganz genau wissen, wonach wir eigentlich suchen, verwenden wir das allgemeine Suchfeld (Search). Nach Eingabe des allgemeinen Suchwortes wasser finden wir in Abbildung 2.5 sowohl die Kronkorken, die als Getränk Wasser eingetragen haben, als auch diejenigen, bei denen dort Mineralwasser steht. Allerdings findet der Suchalgorithmus, der nur untersucht, ob irgendwo in den Daten des Kronkorken die Zeichenkette wasser vorkommt – Groß- bzw. Kleinschreibung ist dabei egal – natürlich beispielsweise auch mehrere Sorten Alsterwasser und Kronkorken der Löwenbrauerei Wasseralfingen.

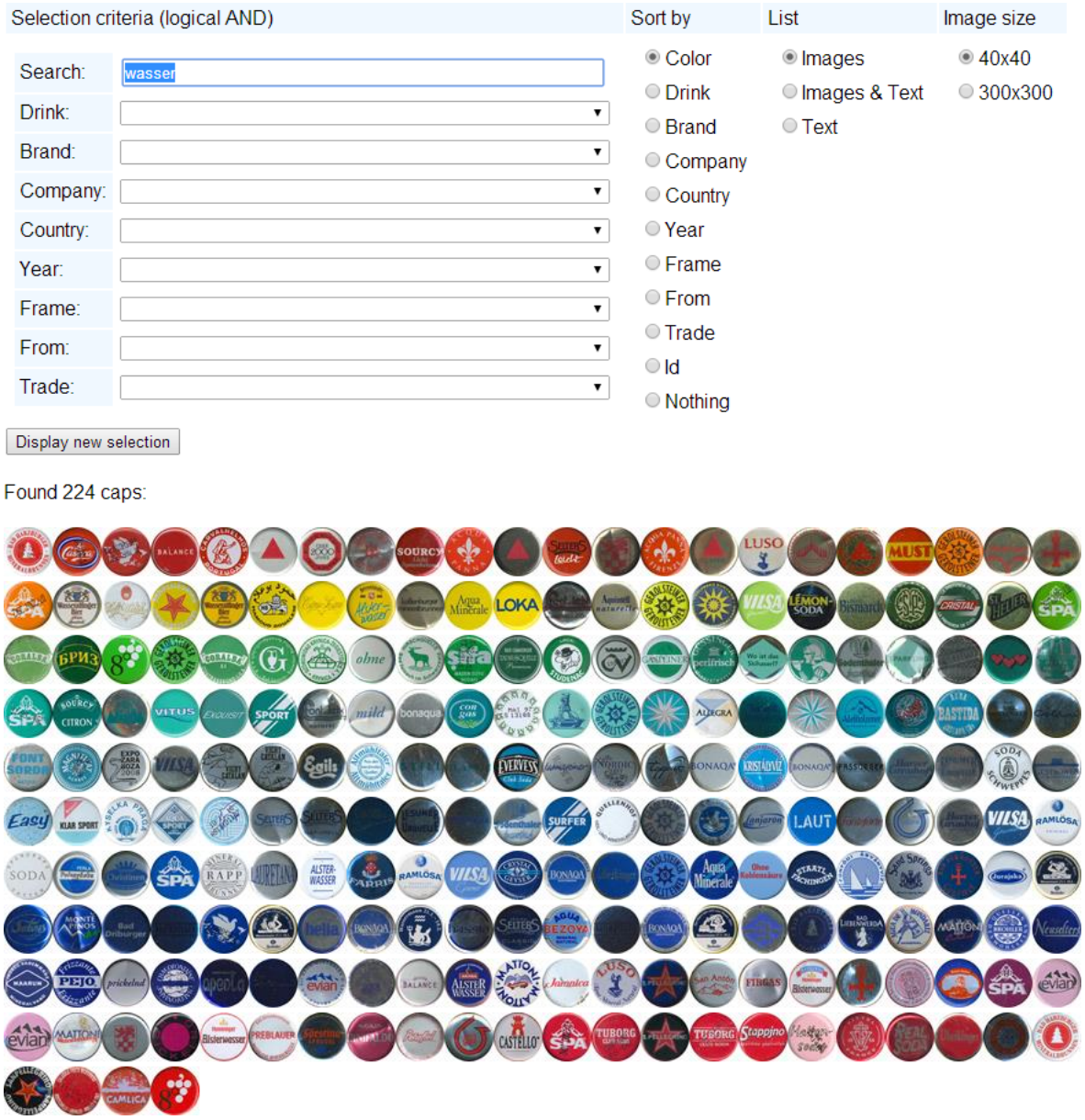

Abbildung 2.5: Wasser (und Mineralwasser und Alsterwasser und . . .)

2.3 Sortierreihenfolge

Nach der Auswahl, welche Kronkorken dargestellt werden sollen, können wir in der zweiten Spalte in Abbildung 2.1 angeben, nach welchen Kriterien die Kronkorken sortiert werden sollen. Wählen wir dort beispielsweise statt der Standardeinstellung (Color) als Sortierkriterium den Hersteller (Company) des Kronkorkens, so sehen wir in Abbildung 2.6, dass die vier Warsteiner-Kronkorken jetzt tatsächlich gemeinsam am Ende zu finden sind, während sie in Abbildung 2.1 wegen ihrer ähnlichen Farbe zwar nahe

beieinander lagen, aber eben doch nicht direkt hintereinander.

Abbildung 2.6: Sortiert nach Hersteller

Bei Auswahl von Nothing als Sortierreihenfolge werden die Kronkorken chronologisch dargestellt. Die zuletzt erfassten Kronkorken lassen sich dann am Ende der Liste leichter auffinden.

2.4 Bild oder Text?

Mit der Standardeinstellung (Images) der dritten Spalte (List) in Abbildung 2.1 sehen wir eine endlose Reihe kleiner Kronkorkenbilder der Größe 40 × 40 Bildpunkte. Wenn wir dort Images & Text auswählen, erhalten wir eine Tabelle, in der neben dem Bild des Kronkorkens auch direkt seine Daten aufgelistet sind (Abbildung 2.7).[4]

Edit	Drink	Brand	Company	Country	Year	Frame	From	Trade	Id
	Bier Alkoholfrei	Holsten Alkoholfrei	Holsten Brauerei AG	Deutschland	2005	Küche	?	0	3f05f67f-83c4-40f5-9943-ddc7ba4d3dff
	Bier Alkoholfrei	Beck's Alkoholfrei	Brauerei Beck GmbH & Co KG	Deutschland	2006	Gäste	?	5	1516d0d1-48bf-4608-becb-fe92e0ff78d9
	Bier Alkoholfrei	früh Kölsch 0,0 %	Cölner Hofbräu	Deutschland	?	Gäste 6	Eberhard Modler	0	1c85cf2a-afed-479c-aab7-6d0cf980bc27

Abbildung 2.7: Bild und Text

Als dritte Variante können wir reinen Text auswählen. Dies kann dann sinnvoll sein, wenn wir eine sehr große Anzahl von Kronkorken herausgefiltert haben, unsere Datenleitung oder der Renderrechner aber etwas schwächeln, da in diesem Fall die Bilder der Kronkorken weder übertragen noch dargestellt werden müssen. Statt durch sein Bild wird dann in Abbildung 2.8 jeder Kronkorken durch einen kleinen blauen Punkt symbolisiert.[5]

[4] In Abbildung 2.7 haben wir das Browserfenster in seiner Breite ziemlich stark zusammen gedrückt. Auf einem großen Monitor sieht die Tabelle mit dann einzeiligen Einträgen wesentlich übersichtlicher aus.

[5] Das Darstellen des immer gleichen blauen Punktes bereitet auch langsamen Browsern kein Problem.

Edit	Drink	Brand	Company	Country	Year	Frame	From	Trade	Id
	Bier Alkoholfrei	Holsten Alkoholfrei	Holsten Brauerei AG	Deutschland	2005	Küche	?	0	3f05f67f-83c4-40f5-9943-ddc7ba4d3dff
	Bier Alkoholfrei	Beck's Alkoholfrei	Brauerei Beck GmbH & Co KG	Deutschland	2006	Gäste	?	5	1516d0d1-48bf-4608-becb-fe92e0ff78d9
	Bier Alkoholfrei	früh Kölsch 0,0 %	Cölner Hofbräu	Deutschland	?	Gäste 6	Eberhard Modler	0	1c85cf2a-afed-479c-aab7-6d0cf980bc27

Abbildung 2.8: Nur Text

Das Anklicken eines Kronkorkenbildes oder blauen Punktes führt uns auch in der Listenansicht zu den Details des jeweiligen Kronkorkens (Kapitel 3).

2.5 Bildgröße

Üblicherweise lassen wir uns kleine Bilder anzeigen; die Image size in der vierten Spalte von Abbildung 2.1 steht dabei also auf 40 × 40 Bildpunkte. Die Größe jedes Bildes beträgt dann ein bis zwei Kilobyte, was übertragungs- und darstellungsseitig auch bei einer großen Anzahl von Bildern noch vertretbar ist. Wenn wir nur sehr wenige Kronkorken herausgefiltert haben, kann es auch sinnvoll sein, in der vierten Spalte der Übersichtsseite als Bildgröße 300 × 300 Bildpunkte auszuwählen. Abbildung 2.9 zeigt die großen Bilder der vier gefundenen Kronkorken. Die Bildgröße verzehnfacht sich bei 300 × 300 Bildpunkten in etwa, die Übertragung und Darstellung dauert entsprechend länger und die Übersichtlichkeit auf einem kleinen Monitor nimmt bei einer großen Anzahl von Bildern natürlich ab.

Abbildung 2.9: Bildgröße 300 × 300 Bildpunkte

3 Detailseite

Durch Anklicken eines Kronkorkens auf der Übersichtsseite gelangen wir auf die in Abbildung 3.1 dargestellte Detailseite.

Drink:	Bier Alkoholfrei	Bier Alkoholfrei
Brand:	Warsteiner Alkoholfrei	Warsteiner Alkoholfrei
Company:	Warsteiner Brauerei Haus Cramer GmbH & Co KG	Warsteiner Brauerei Haus Cramer GmbH & Co KG
Country	Deutschland	Deutschland
Year:	2005	2005
Frame:	Küche	
From:	?	
Trade:	0	
Id:	e7c3be0c-9b66-40a4-8962-a1e6e41596a6	
IP:	91.55.127.39	
Comment:	Dieser Vorschlag ist natürlich nur ein Test.	

Send correction suggestion | Back to the previous page

Login

Abbildung 3.1: Detaildarstellung eines Kronkorkens

Alternativ können wir – wenn wir die Identifikationsnummer (Id) eines Kronkorkens kennen, diese direkt beim Aufruf der Detailseite angeben:

`http://suse.thebuchis.de/crown/cap.aspx?id=e7c3be0c-9b66-40a4-8962-a1e6e41596a6`

Auf der Detailseite sind neben einem großen Bild (300 × 300 Bildpunkte) des Kronkorken alle seine Daten übersichtlich aufgelistet. Zusätzlich zu den im Tooltip auf der Übersichtsseite dargestellten Informationen wird auf der Detailseite noch die IP-Adresse des aktuellen Nutzers ausgegeben und es gibt ein Kommentarfeld, das der Nutzer bei einem Korrekturvorschlag (Abschnitt 3.1) verwenden kann.

3.1 Korrekturvorschlag

Wenn ein Nutzer einen Fehler in den Daten eines Kronkorken entdeckt oder etwas über den Kronkorken weiß, das wir nicht wissen, kann er uns direkt auf der Detailseite einen Vorschlag für eine Korrektur bzw. Ergänzung machen. Dazu trägt er seine Korrekturen einfach in die Felder Drink, Brand, Company, Country und/oder Year ein, schreibt gegebenenfalls noch einen kurzen Kommentar dazu und drückt die Send correction suggestion Schaltfläche. Die übrigen Felder (Frame, ...) sind für ihn gesperrt, da es ziemlich unwahrscheinlich ist, dass ein externer Nutzer beispielsweise weiß, in welchem Rahmen sich der Kronkorken befindet.☺

Rechts neben den Eingabefeldern befindet sich jeweils eine Auswahlliste, mit der der Nutzer die schon vorhandenen Einträge in die Eingabefelder kopieren kann, um Doppeleingaben zu vermeiden.

Der Korrekturvorschlag wird uns dann als die in Abbildung 3.2 dargestellte E-Mail geschickt, in der die zur Korrektur vorgeschlagenen Einträge besonders markiert sind. Ein Anklicken des Kronkorkenbildes in der E-Mail öffnet direkt die Detailseite im Browser, so dass wir die Änderungen dort gegebenenfalls sofort umsetzen können.

Abbildung 3.2: E-Mail eines (nicht ernst gemeinten) Korrekturvorschlags

3.2 Zurück

Die Back to the previous page Schaltfläche ruft die letzte im Browsercache gespeicherte Seite wieder auf, was dem Drücken des Zurück Pfeils des Browsers entspricht. Dieses Zurückblättern im Browser hat den großen Vorteil, dass die Kronkorkenbilder nicht noch einmal geladen werden müssen, was der Fall wäre, wenn wir die Übersichtsseite direkt erneut aufrufen würden. Je nachdem wie viele Aktionen wir gerade auf der Detailseite durchgeführt haben, kann es allerdings notwendig sein, die Schaltfläche mehrmals zu drücken, da jedes Mal nur eine Aktion zurückgeblättert wird. Alternativ können wir in allen modernen Browsern den Zurück Pfeil etwas länger drücken; wir bekommen dann eine Liste der letzten Seiten angezeigt und können direkt zur Übersichtsseite zurück springen.

3.3 Login

Um die Daten eines Kronkorkens ändern zu können, müssen wir auf der Detailseite ganz unten das für externe Nutzer natürlich unbekannte Kennwort eintragen und die Login Schaltfläche drücken. Wenn das Anmelden geklappt hat, werden die vorher gesperrten Felder Frame, From und Trade freigegeben und es erscheint eine Änderungen abspeichern

und eine Diesen Kronkorken ohne Sicherheitsabfrage löschen Schaltfläche. Außerdem wird eine Zurück zur Auswahl Schaltfläche sichtbar gemacht, die wir zur in Kapitel 4 beschriebenen Kronkorkenerfassung verwenden werden.

4 Kronkorkenerfassung

Die Erfassung neuer Kronkorken ist für einen externen Nutzer nicht möglich. Trotzdem wollen wir das Verfahren hier beschreiben, da es möglicherweise andere Nutzer anregen könnte, etwas Ähnliches zu programmieren.

4.1 Einscannen

Um einen neuen Kronkorken in die Datenbank einpflegen zu können, müssen wir natürlich erst einmal sein Bild aufnehmen. Dabei bieten sich zwei Methoden an: Fotografieren oder Einscannen. Da mittlerweile praktisch jedes Mobiltelefon eine vernünftige Kamera besitzt, erscheint es am zweckmäßigsten, den neuen Kandidaten einfach zu fotografieren. Dies hat allerdings zwei entscheidende Nachteile: Erstens sind die Lichtverhältnisse immer wieder unterschiedlich,[1] so dass auch die Farben des Kronkorkens auf verschiedenen Aufnahmen variieren, was die spätere Dublettenerkennung erschwert und zweitens ist es ohne technischen Aufwand aus der Hand praktisch unmöglich, den Aufnahmewinkel und die Größe des Kronkorken (und damit seine Bildpunkteanzahl) auf allen Aufnahmen gleich zu halten.

Mit einem Flachbettscanner wiederum können wir beide Probleme leicht lösen. Der Scanner erzeugt seine eigene, immer gleiche Beleuchtung, seine Auflösung können wir bildpunktgenau einstellen und die Kronkorken liegen flach und eben[2] auf dem Glas auf.

Der Innendurchmesser und der Durchmesser des Aufdrucks eines Kronkorkens betragen ziemlich genau ein Zoll (1" = 25.4 mm), so dass eine am Scanner häufig direkt auswählbare Auflösung von 300 dpi[3] praktischerweise genau zum Bild eines einzelnen Kronkorkens von 300×300 Bildpunkten führt.

Bei klassischer Wabenpackung (abwechselnd Siebener- und Sechserreihen mit ausreichend Randabstand) können wir 72 Kronkorken auf einer DIN A4-Scannerseite unterbringen. In Abbildung 4.1 erkennen wir ein Problem vieler handelsüblicher Scanner: Durch die mittige Anordnung des optischen Sensors werden am linken und rechten Rand liegende Kronkorken etwas schräg von der Seite eingescannt.

[1] Ein Blitz würde bei vielen glänzenden Kronkorken unerwünschte Reflexionen erzeugen.

[2] Sehr stark geknickte Kronkorken sollten wir vor dem Scannen möglichst platt drücken, da manche Scanner – konstruiert für zweidimensionale Papierseiten – nur eine sehr geringe Tiefenschärfe besitzen und die nicht aufliegenden Bereiche des Kronkorkens unscharf abbilden.

[3] 300 dpi bedeutet 300 dots per inch, also 300 Bildpunkte pro Zoll.

Abbildung 4.1: Scan von 72 Kronkorken

Dieser sogenannte Parallaxenfehler hat für diese außen liegenden Kronkorken zur Folge, dass nicht nur die gezackten Ränder, sondern auch der gewölbte Teil des Aufdrucks, der nicht plan aufliegt, außen nicht vernünftig abgebildet werden. Da wir aber im Folgenden nur den inneren, ungewölbten Teil des Aufdrucks verwenden werden, ist dieses Problem nicht sehr relevant. [4]

Ein weiteres Problem vieler Scanner stellt die Richtung dar, aus der die Kronkorken während des Scanvorganges beleuchtet werden. In Abbildung 4.1 sehen wir sehr deutlich, dass das Licht auf alle Kronkorken von unten fällt. Besonders die Kronkorken mit

[4] Echte Hardcorekronkorkensammler unterscheiden ihre Kronkorken auch noch hinsichtlich der Randzeichen, auf denen die Druckereien ihre jeweiligen Insignien hinterlassen. Wir verzichten auf diese Unterscheidung; für uns ist nur der plane Aufdruck des Kronkorkens entscheidend.

metallischer Oberfläche zeigen deutlich leuchtende „Möndchen" am unteren Rand. Da wir alle Kronkorken später in ihre Normalform drehen werden, damit beispielsweise die Schrift normal lesbar ist, könnte es passieren, dass der gleiche Kronkorken einmal mit einem Möndchen unten und ein zweites Mal mit einem Möndchen oben eingescannt wird, was die Dublettenerkennung natürlich erschwert. Dieses Problem werden wir lösen, indem wir für die Dublettenerkennung nicht den gesamten Aufdruck, sondern nur einen kleineren, inneren Bereich verwenden, in dem die Möndchen nicht enthalten sind.

Generell ist die Dublettenerkennung bei Kronkorken mit metallisch spiegelnden Oberflächen natürlich problematisch. Die Erkennung funktioniert ja über Farbvergleiche und welche Farbe hat denn eigentlich ein Spiegel? Wir müssen also damit rechnen, dass die Dublettenerkennungsrate bei „silbernen und goldenen" Kronkorken deutlich abnimmt.

4.2 Hochladen

Bevor wir einzelne Kronkorken erfassen können, müssen wir den Scan auf den Server hochladen. Dies geschieht auf der Seite `kronkorken_auswaehlen.aspx`, auf der wir uns anmelden (Abbildung 4.2a), um den Scan auf unserem Rechner auszuwählen und hochzuladen (Abbildung 4.2b).

(a) Nicht angemeldet (b) Angemeldet

Abbildung 4.2: Seite `kronkorken_auswaehlen.aspx`

Nach dem Hochladen des Scans erscheint dann das in Abbildung 4.1 dargestellte Bild auf der Seite.

4.3 Ausrichten

Wenn wir dort jetzt in etwa auf die Mitte eines einzelnen Kronkorkens klicken – beispielsweise auf den hellgrünen Gerolsteiner – wird auf der nächsten Webseite (`ausschnitt_bestimmen.aspx`) ein 400×400 Bildpunkte großer Detailausschnitt dargestellt (Abbildung 4.3).[5]

[5] Um im nächsten Schritt einzelne Punkte präziser anklicken zu können, wird die Darstellungsauflösung dabei auf 800×800 Bildpunkte aufgeblasen.

Abbildung 4.3: Vergrößerter Detailausschnitt des ausgewählten Kronkorkens

In diesem Detailausschnitt müssen wir nun genau den Mittelpunkt des Kronkorkens finden und ihn dann so in seine Normalform drehen, dass die Schrift lesbar ist.

4.3.1 Ausschnitt bestimmen

Kronkorken sind rund. Deshalb lassen sich auch in den Aufdrucken eines Großteils der Kronkorken konzentrische Objekte (Kreise, Sterne, ...) finden, deren Mittelpunkt mit dem Mittelpunkt des Kronkorken übereinstimmt. Im Aufdruck des Gerolsteiners aus Abbildung 4.3 lassen sich beispielsweise gleich mehrere Kreise mit dem gleichen Mittelpunkt finden: der äußere schwarze Kreis, der untere Rand der Schrift, der obere Rand der Schrift, die äußeren Spitzen des Sterns und der innere Kreis im Stern.

Um nun den kreisförmigen Ausschnitt mit dem Durchmesser von 300 Bildpunkten zu definieren, brauchen wir seinen Mittelpunkt. Da wir durch drei Punkte immer genau einen Kreis legen können, klicken wir in Abbildung 4.3 drei Punkte auf dem äußeren schwarzen Kreis an. Das Programm berechnet aus diesen drei Punkten – in Abbildung 4.4 in weiß dargestellt – den Mittelpunkt und blendet dann alles außerhalb des 300er-Kreises aus.

Je weiter die einzelnen Punkte voneinander entfernt sind, desto genauer wird der Mittelpunkt gefunden. Es ist also sinnvoll, die Punkte möglichst nahe am Rand des Kronkorkens zu setzen und dann möglichst in Form eines gleichseitigen Dreiecks zu verteilen.

Abbildung 4.4: Drei Klicks (in weiß) definieren den Mittelpunkt.

4.3.2 Drehen

Jetzt müssen wir den Kronkorken in seine Normalform drehen, sodass die Schrift lesbar wird. Dazu definieren wir durch zwei Klicks – in Abbildung 4.4 in rot dargestellt – eine (später) waagerechte Linie, die das Programm verwendet, um den Kronkorken auszurichten. Während die Reihenfolge der Klicks bei der Kreisbestimmung in Abschnitt 4.3.1 egal war, ist die Reihenfolge der Klicks bei der Horizontbestimmung entscheidend. Der erste Punkt (in Abbildung 4.4 das rechte, untere, rote Kreuz) definiert den Anfang der horizontalen Strecke, der im gedrehten Bild später links liegt.

Die waagerechte Linie muss dabei nicht durch den Mittelpunkt des Kronkorkens verlaufen; sie kann auch in der oberen oder unteren Hälfte liegen. Wie auch schon bei der Kreisbestimmung wird die Drehung des Kronkorkens desto präziser, je größer die Entfernung zwischen den beiden Punkten ist. Auch hier müssen wir häufig einen Kompromiss eingehen, da wir waagerechte Punkte nicht immer auf dem Rand des Kronkorkens finden.

Auch bei rotationssymmetrischen oder gar einfarbigen Kronkorken (Abbildung 4.5a) müssen wir aus formalen Gründen beide Horizontalpunkte eingeben; ihre Lage ist dabei aber natürlich egal.

(a) Rotationssymmetrisch oder einfarbig

(b) Keine eindeutige Ausrichtung

Abbildung 4.5: Probleme bei der Horizontbestimmung

Auf manchen Kronkorken (Abbildung 4.5b) ist es überhaupt nicht möglich, eine eindeutige horizontale Ausrichtung zu erkennen, wenn beispielsweise Schriften in unterschiedlichen Winkeln auftreten. Wir können dann nur versuchen, selbst eindeutige Regeln („Bannerschriften nicht zur Ausrichtung verwenden", ...) zu definieren und uns möglichst konsequent daran zu halten.

4.3.3 Feinkorrektur

Nachdem das Programm den Kronkorken gedreht hat, haben wir die Möglichkeit, ihn noch punktgenau auszurichten, bevor er an die Dublettenerkennung übergeben wird. Dabei haben die klassischen „Gamer"-Tasten die folgende Wirkung:

- **w** Kronkorken nach oben verschieben
- **a** Kronkorken nach links verschieben
- **s** Kronkorken nach unten verschieben
- **d** Kronkorken nach rechts verschieben
- **q** Kronkorken links herum drehen
- **e** Kronkorken rechts herum drehen
- **g** Gitter einschalten

Wenn wir das Gitter mit der g-Taste eingeschaltet haben (Abbildung 4.6), können wir an den senk- und waagerechten Linien und den konzentrischen Kreisen sehr genau erkennen, ob der Kronkorken schon exakt zentriert und gedreht ist, oder ob wir noch etwas nachjustieren müssen.

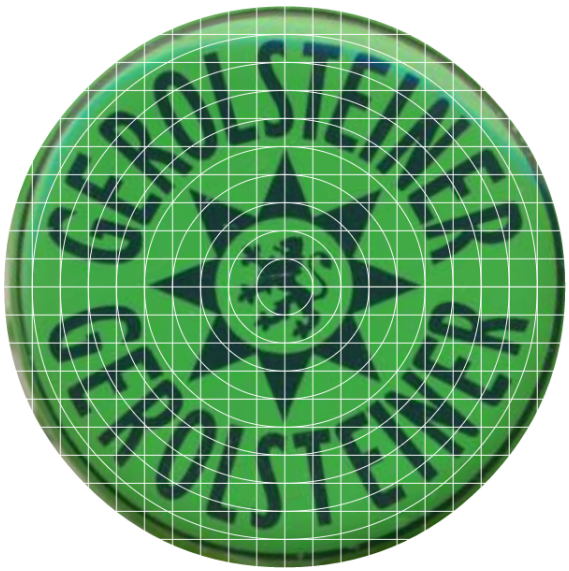

Abbildung 4.6: Feinkorrektur

Nach dem fünften Klick (letzter Klick der Horizontbestimmung) blenden wir die Schaltfläche Vergleich mit vorhandenen Kronkorken ein. Durch das Drücken der Schaltfläche gelangen wir auf die Webseite `vergleichen.aspx`, auf der der aktuelle Kronkorken mit allen in der Datenbank vorhandenen verglichen wird, um mögliche Dubletten zu finden.

4.4 Dublettenerkennung

Wenn wir mehr als ein paar hundert Kronkorken haben, wird es zunehmend schwieriger zu wissen, ob wir einen neuen Kandidaten schon besitzen. Das Programm vergleicht daher das Bild des Kandidaten mit allen schon in der Datenbank vorhandenen Kronkorken und zeigt uns die Kronkorken, die dem Kandidaten am ähnlichsten sind.

4.4.1 Verringerung der Bildpunkte

Dazu verringern wir die Auflösung des Kandidaten von 300×300 auf 40×40 Bildpunkte (Abbildung 4.7). Dies hat auf der einen Seite den Vorteil, dass wir deutlich weniger Bildpunkte zum Vergleich heranziehen müssen;[6] auf der anderen Seite bilden wir bei der Auflösungsverringerung ja immer die Mittelwerte mehrerer Originalbildpunkte und verringern so die Auswirkungen kleiner Verschiebungs- oder Drehungsfehler bei sehr detailreichen Kronkorken.

[6] Der Einsparungsfaktor ist tatsächlich sehr hoch: $\frac{300 \cdot 300}{40 \cdot 40} = 56,25$. Und wenn wir dann noch bedenken, dass jeder Kandidat mit vielen tausend anderen Kronkorken verglichen wird, ...

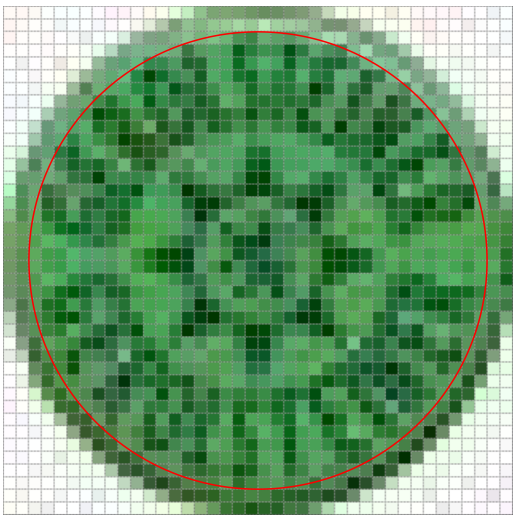

Abbildung 4.7: Herunter skaliert auf 40 × 40 Bildpunkte

Des Weiteren verwenden wir zur Dublettenerkennung nur die Bildpunkte, die im Inneren des in Abbildung 4.7 rot dargestellten Kreises mit einem Durchmesser von 36 Bildpunkten liegen. Dies verhindert – neben der weiteren Verringerung der zu vergleichenden Bildpunkte – dass die in Abbildung 4.7 deutlich sichtbaren, als Kompressionsartefakte bezeichneten, zufälligen Farbunterschiede im eigentlich weißen Randbereich die Dublettenerkennung erschweren. Außerdem werden durch die Beschränkung auf den inneren Kreis die in Abschnitt 4.1 angesprochenen Möndchen, die durch die seitliche Scannerbelichtung entstehen und die in Abbildung 4.3 gut erkennbar sind, weitestgehend ausgeblendet.

Da wir die 40 × 40 Bilder ja auch für die in Kapitel 2 beschriebene Übersichtsseite benötigen, speichern wir von jedem Kronkorken ein Bild mit 300×300 und eins mit 40×40 Bildpunkten auf dem Server ab. Der Dublettenerkennungsalgorithmus kann jetzt also jeden einzelnen Bildpunkt des inneren 36er-Kreises des Kandidaten mit jedem einzelnen Bildpunkt des inneren Kreises aller schon vorhandenen Kronkorken vergleichen und uns die ähnlichsten Exemplare zur weiteren Begutachtung vorlegen (Abbildung 4.8).[7]

[7] In der Praxis sehen wir nicht nur die ersten 16, sondern die 100 ähnlichsten Kronkorken. Diese werden zwar in der Auflösung 300×300 angefordert, um Platz zu sparen aber nur mit 100×100 Bildpunkten dargestellt.

Abbildung 4.8: Dublettenerkennung (Farbige Kronkorken)

4.4.2 Farbe

Mustererkennungsalgorithmen werden seit ein paar Jahrzehnten in sehr vielen Bereichen unseres Lebens eingesetzt (Schrifterkennung, Spracherkennung, Gesichtserkennung, Situationserkennung, Fehlererkennung, ...) und auf ihre jeweilige Anwendung hin optimiert. In unserem Fall, in dem die zu erkennenden Dubletten den Kandidaten durch die genaue Ausrichtung und die reproduzierbaren Randbedingungen des Scanners sehr ähnlich sehen, reicht es den meisten Fällen, direkt die Farben entsprechender Bildpunkte miteinander zu vergleichen.

Dabei ist der Begriff „Farbe" allerdings nicht eindeutig definiert. Eine übliche Farbbeschreibung verwendet die Zerlegung der Farbe eines Bildpunktes in die Anteile der drei Grundfarben Rot, Grün und Blau (RGB-Farbraum). Alternativ können wir die Farbe jedes Bildpunktes beispielsweise auch mit den drei Größen Farbton (englisch: Hue), Farbsättigung (englisch: Saturation) und Hellwert (englisch: Value) beschreiben (HSV-Farbraum).

Je nachdem, welche Farbbeschreibungswerte wir für den Vergleich verwenden, erhalten wir teilweise recht unterschiedliche Erkennungsergebnisse: In Abbildung 4.8 verwenden wir nur die Farbtöne (Hue) der jeweiligen Bildpunkte. Es werden dann überwiegend hellere, farbkräftige Kronkorken gleicher Farbe gefunden. Da der schon vorhandene Gerol-

steiner allerdings einen signifikant abweichenden Farbton hat (sein Grün hat, vermutlich druckbedingt, einen deutlich gelberen Farbton und er zeigt aufgrund eines Knicks starke Reflexionen), wird er „erst" an dreizehnter Stelle gefunden.[8]

Abbildung 4.9 verwendet statt des Farbtones alle drei RGB-Werte zum Vergleich und liefert üblicherweise dunklere, weniger farbenfreudige Kronkorken, auf denen eher auch gänzlich andere Farben auftreten (rote und blaue Schrift). Die gesuchte Dublette finden wir hier an erster Stelle.

Abbildung 4.9: Dublettenerkennung (Graue Kronkorken)

Auch in den beiden Teilen von Abbildung 4.10 offenbaren die beiden Verfahren ihre Unterschiede.

[8] Meistens führt die Verwendung des Farbtones statt des RGB-Wertes allerdings zu höheren Erkennungserfolgen.

(a) Auswahl: Farbige Kronkorken (b) Auswahl: Graue Kronkorken

Abbildung 4.10: Unterschiedliche Algorithmen führen zu unterschiedlichen Erkennungsergebnissen

Links steht der Mittelwert des Farbtones des Kronkorkens im Vordergrund. Es werden also überwiegend kräftige Orange- und Brauntöne gefunden. Rechts spielen auch Sättigung und Helligkeit eine Rolle. Dies führt dazu, dass sehr schnell fast alle Kronkorken der Aloha-Marke mit ähnlicher Farbe gefunden werden und im weiteren Verlauf (an 20. und 37. Stelle) interessanterweise sogar noch Aloha-Kronkorken, die sich sowohl hinsichtlich der Farbe und der Größe der Blume als auch der Hintergrundfarbe recht stark vom Kandidaten unterscheiden. In beiden Fällen finden wir die gesuchte Grapefruit-Dublette an erster Stelle.

Noch deutlicher wird der Unterschied zwischen den beiden Vergleichsverfahren in Abbildung 4.11.

(a) Dimix: Farbige Kronkorken (b) Dimix: Graue Kronkorken

Abbildung 4.11: Suche nach Dimix-Dublette

Wenn wir hier nur den Farbton (Hue) berücksichtigen (Abbildung 4.11a), finden wir – neben der gesuchten Dublette an erster Stelle – auch Kronkorken, die zwar den gleichen Farbton besitzen, aber beispielsweise (Granini) viel farbintensiver und heller sind. Obwohl der Kandidat in diesem Fall deutlich dunkler als die Dublette ist (vgl. Abbildung 4.11a), wird er gefunden, da er den gleichen Farbton besitzt.[9]

In Abbildung 4.11b verwenden wir alle drei Grundfarben (RGB) und damit neben dem Farbton auch die Sättigung und den Hellwert zum Vergleich, so dass hier nur Korken gefunden werden, die dem Kandidaten in allen drei Merkmalen ähnlich sind. Es werden daher im Vergleich mit 4.11a überwiegend dunkelrote (und „goldene") Kronkorken gefunden. Da Kandidat und gesuchte Dublette aber stark unterschiedliche Helligkeiten besitzen, wird die Dublette unter den ersten ausgegebenen Exemplaren nicht gefunden.

Es empfiehlt sich daher, im Zweifelsfall unbedingt beide Erkennungsverfahren auszuprobieren. Vermutlich werden im Lauf der zukünftigen Entwicklung des Programms weitere Verfahren implementiert werden, die beispielsweise auch verstärkt die Muster (Schriftzüge, Embleme, ...) zur Dublettenerkennung heranziehen. Spätestens dann wird es auch sinnvoll werden, die von den unterschiedlichen Verfahren gefundenen Exemplare hinsichtlich ihrer Wahrscheinlichkeit zu wichten, um dem Nutzer nur eine gemeinsame Liste auszugeben.

[9] Die Möndchen, die beim Kandidaten unten links und bei der Dublette oben rechts liegen, werden ja gemäß Abschnitt 4.4.1 ausgeblendet und tragen daher nicht zur Unterscheidung bei.

4.4.3 Gefunden?

Wenn wir eine Dublette gefunden haben, klicken wir einfach auf diese und gelangen zu der in Kapitel 3 beschriebenen und in Abbildung 4.12 nochmals dargestellten Detailseite.[10]

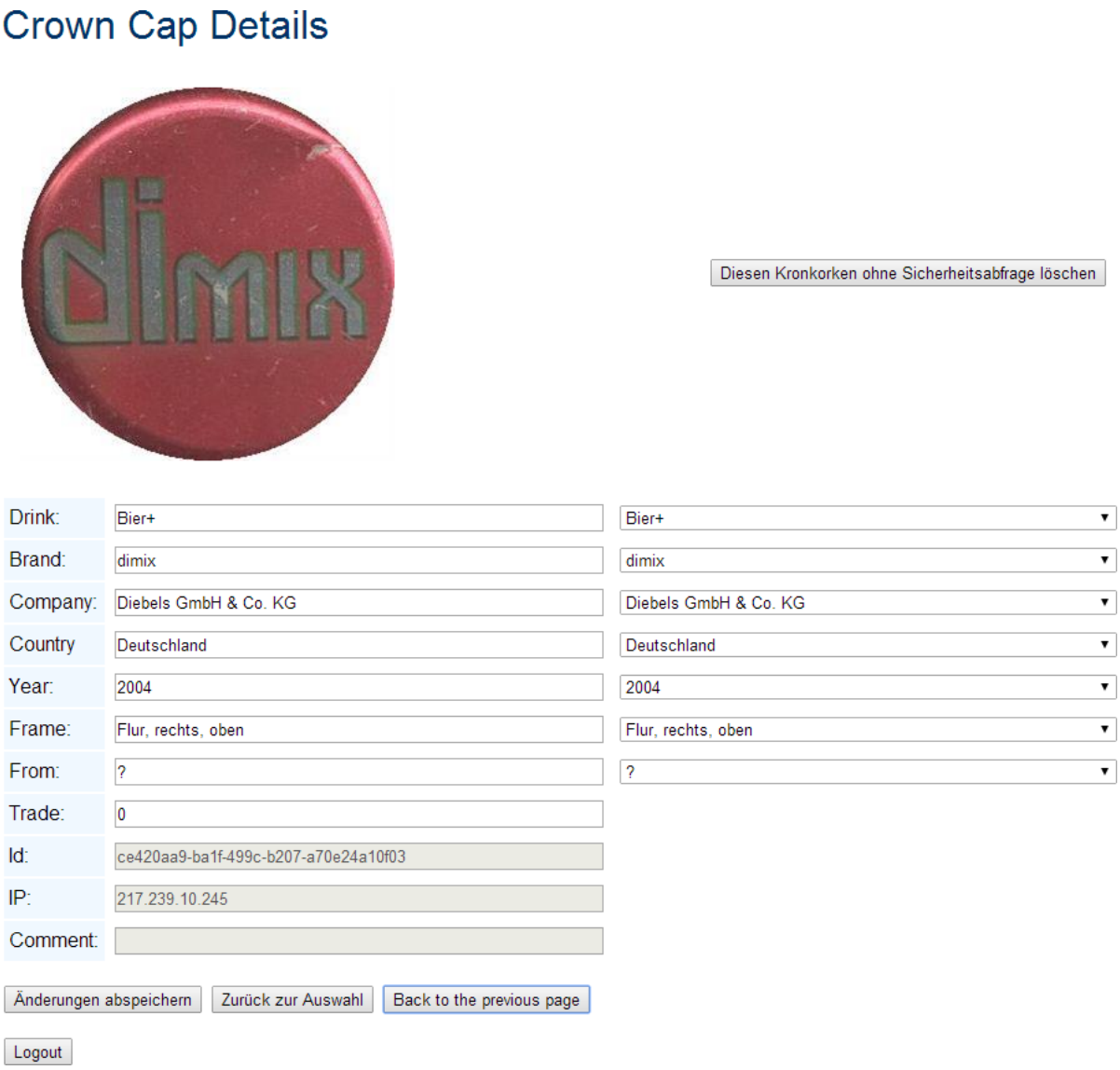

Abbildung 4.12: Detailseite zum Erhöhen des Tauschzählers

Hier können wir jetzt den Tauschzähler (Trade) um eins erhöhen, der anderen Kronkorkensammlern anzeigt, wie viele Kronkorken dieses Typs wir zum Tauschen bevorraten. Nachdem wir mit Änderungen abspeichern den neuen Tauschzähler abgespeichert haben, kehren wir durch Anklicken von Zurück zur Auswahl auf die Auswahlseite zurück,

[10] Hier fällt auf, dass die Dublette nicht exakt horizontal ausgerichtet ist, was die Erkennung natürlich erschwert.

auf der wir in dem in Abbildung 4.1 dargestellten Scan einen neuen Kandidaten auswählen können. Alle schon vorher untersuchten Kandidaten werden dabei mit einem weißen Quadrat gekennzeichnet (Abbildung 4.13).[11]

Abbildung 4.13: Schon untersuchte Kandidaten werden markiert.

Wenn wir für den Kandidaten keine Dublette gefunden haben, fügen wir ihn durch Anklicken der Schaltfläche In die Datenbank aufnehmen in Abbildung 4.8 unserem Bestand hinzu. Auch hierdurch gelangen wir wieder zu der Detailseite (Abbildung 4.12), auf der wir nun die Daten des neuen Kronkorkens eintragen und abspeichern. Dabei ist es sehr sinnvoll, vorher in der Liste der gefundenen Kronkorken ein Exemplar anzuklicken, das möglichst viele gemeinsame Daten mit dem neuen Kandidaten hat, da die Details des zuletzt aufgerufenen Kronkorkens als Startwerte für den neuen Kandidaten eingetragen werden.[12] Wir müssen auf diese Weise dann nur noch die wenigen Daten ändern, in denen sich der Kandidat vom schon in der Datenbank vorhandenen Exemplar unterscheidet.

Um Doppeleinträge mit ähnlicher (aber nicht exakt identischer) Schreibweise zu vermeiden, ist es in den meisten Fällen sehr sinnvoll, in den Auswahllisten der rechten Spalte in Abbildung 4.12 nach schon vorhandenen, passenden Einträgen zu suchen, bevor wir in der linken Spalte einen neuen Eintrag vornehmen.

[11] Die weißen Quadrate sind reine Webseitenoverlays; Das Originalbild und die Kandidaten selbst werden nicht verändert und nach einem erneuten Anmelden sind die Markierungen wieder verschwunden.

[12] Wenn in der Liste der gefundenen Kronkorken kein passendes Exemplar dabei ist, können wir einfach in einem weiteren Browserfenster einen passenden Kronkorken in seiner Detailansicht aufrufen, um die Startwerte des neuen Kandidaten festzulegen.

Teil II

Wie geht'n das?

5 Aufbau

Abbildung 5.1 zeigt ein Übersichtsdiagramm, in dem alle Seiten des Programms und ihre Aufruftopologie dargestellt sind.

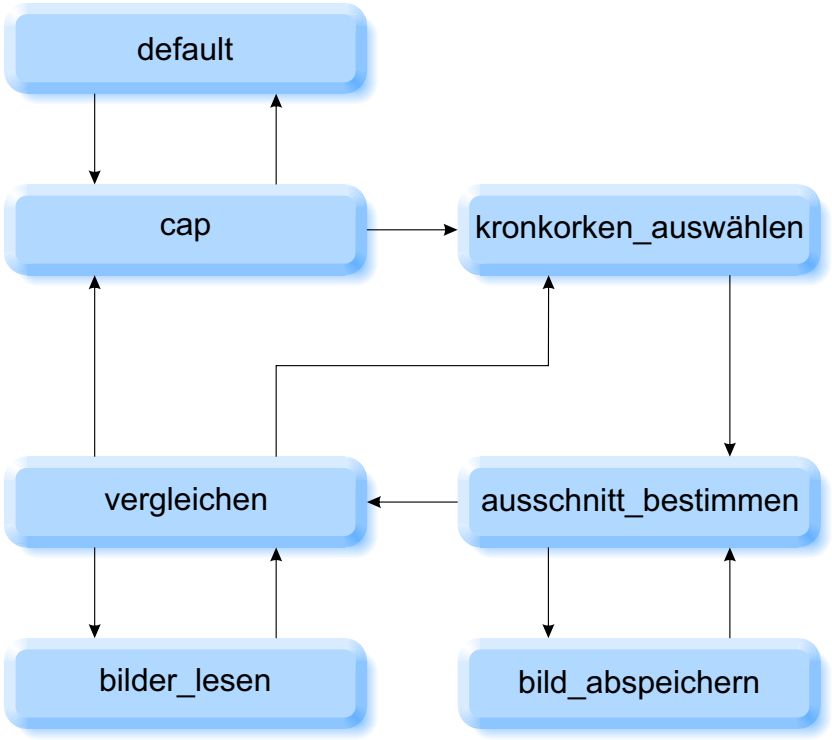

Abbildung 5.1: Blockdiagramm

Die Seite `default.aspx` (Kapitel 7) ist die Einstiegs- und Übersichtsseite in das Programm und stellt die ausgewählten Kronkorken als Bilder und in Listen dar. Durch Anklicken eines Kronkorkens gelangen wir auf seine Detailseite. Dabei übertragen wir die Id des aktuellen Kronkorkens über den QueryString in der URL der Detailseite.

Die Seite `cap.aspx` (Kapitel 3) stellt ein 300 × 300 Bild und die Daten des Kronkorkens dar und ermöglicht es Nutzern, Korrekturvorschläge per E-Mail zu senden. Der Administrator kann sich auf dieser Seite anmelden und dann alle Daten des aktuellen Kronkorkens in der Datenbank ändern oder den Kronkorken löschen. Ein Klick auf die Schaltfläche Back to the previous page führt uns wieder zurück zur Übersichtsseite.

Wenn wir als Administrator angemeldet sind, können wir auf Zurück zur Auswahl klicken, um zur Auswahlseite zu gelangen, auf der wir Kronkorken zur Dublettensuche und gegebenenfalls zur Aufnahme in die Datenbank auswählen können.

Auf der Seite `kronkorken_auswaehlen.aspx` (Kapitel 9) können wir einen Scan von vielen Kronkorken auf den Server hochladen und aus dem Scan einen einzelnen Kronkorken auswählen. Nach Anklicken eines Kronkorkens speichern wir die Koordinaten des Klicks im localStorage-Objekt der Anwendung und wechseln zur nächsten Seite, auf der wir den Ausschnitt des Kronkorkens bestimmen.

Auf der Seite `ausschnitt_bestimmen.aspx` (Kapitel 10) lesen wir die Klickkoordinaten aus dem localStorage-Objekt wieder aus und verwenden fünf weitere Mausklicks, um den genauen Mittelpunkt und die horizontale Ausrichtung des Kronkorkens festzulegen. Anschließend können wir – unterstützt durch ein kartesisches und radiales Gitter – mittels bestimmter Tasten eine exakte Feinjustierung der Lage und Ausrichtung des Ausschnittes vornehmen. Ein Klick auf Vergleich mit vorhandenen Kronkorken sendet das fertige Bild an einen Webdienst auf dem Server.

Auf der Serverseite nimmt der Webdienst `bild_abspeichern.aspx` (Kapitel 11) das Bild entgegen und speichert es als 300×300 und als 40×40 Bild ab. Wenn das Abspeichern erfolgreich war, übergibt der Webdienst die Kontrolle wieder zurück an die Seite `ausschnitt_bestimmen.aspx`, die dann die Vergleichsseite aufruft, auf der wir mögliche Dubletten suchen.

Auf der Seite `vergleichen.aspx` (Kapitel 12) vergleichen wir den aktuellen Kandidaten mit allen in der Datenbank vorhandenen Kronkorken und stellen die 100 ähnlichsten Dublettenkandidaten dar. Dazu benutzen wir beim ersten Aufruf der Seite die Hilfsseite `bilder_lesen.aspx` (Kapitel 13), um alle schon vorhandenen 40×40 Kronkorkenbilder einzulesen und – zusammen mit ihren Ids – in einem Bytefeld abzuspeichern. Von der Vergleichsseite gelangen wir – wenn wir keine Dublette gefunden haben und den Kandidaten neu in die Datenbank aufnehmen wollen – durch Drücken der Schaltfläche In die Datenbank aufnehmen auf die Detailseite, um dort die Eigenschaften des neuen Kronkorkens einzutragen. Wenn aber der Kandidat offensichtlich schon in der Datenbank vorhanden ist und wir eine Dublette gefunden habe, führt uns ein Klick auf Zurück zur Auswahl wieder zurück zur Auswahlseite.

6 kronkorken.xml

Wir speichern die Daten (`id`, `getraenk`, ...) aller Kronkorken gemeinsam in einer XML-Datenbank ab:[1]

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <kronkorken>
    <id>42</id>
    <getraenk></getraenk>
    <marke></marke>
    <hersteller></hersteller>
    <land></land>
    <jahr></jahr>
    <rahmen></rahmen>
    <farbe></farbe>
    <tauschen></tauschen>
    <von></von>
  </kronkorken>
  <kronkorken>
    <id>b1c78453-c112-4dc7-9b56-68533d379949</id>
    <getraenk>Brause</getraenk>
    <marke>Moxie Original Elixir</marke>
    <hersteller>Mr. Yin and Mr. Yang, LLC</hersteller>
    <land>USA</land>
    <jahr>2003</jahr>
    <rahmen>Flur, Mitte rechts, oben</rahmen>
    <farbe>197</farbe>
    <tauschen>0</tauschen>
    <von>?</von>
  </kronkorken>
  :
</root>
```

XML-Datenbanken haben den großen Vorteil, dass sie in direkt lesbaren Textdateien abgespeichert werden,[2] kein externes Datenbankmanagementsystem benötigen und von

[1] Aus Übersichtlichkeitsgründen: Ein einzelner Doppelpunkt in einer Programmierzeile steht hier und im Folgenden für (beliebig viele) weitere gleichartige Objekte.

[2] Auf diese Weise könnten wir – wenn wir beispielsweise einen Rahmen in ein anderes Zimmer umhängen – die Rahmennamen aller zugehörigen Kronkorken sehr einfach direkt in der XML-Datei mittels Suchen und Ersetzen korrigieren.

VisualBasic aus mittels LINQ (Language Integrated Query) sehr komfortabel angesprochen und verarbeitet werden können.

Der allererste Kronkorken in der Datei hat eine `id` von `42` (statt der automatisch generierten, langen `id`s der übrigen Kronkorken) und ansonsten nur leere Einträge (`getraenk`, `marke`, ...). Er wird benötigt, damit wir in den automatisch aus der XML-Datei erzeugten Auswahllisten auf der Übersichtsseite (Abbildung 2.1) auch leere Einträge anwählen können.

7 default.aspx

Die Seite `default.aspx` ist die Einstiegsseite (Kapitel 2) in das Programm und stellt die ausgewählten Kronkorken als Bilder und in Listen dar.

7.1 Page_Load

Das „Hauptprogramm" `Page_Load` füllt die Auswahllisten mit den Daten aus der Datenbank (`initialisierung`), setzt den Cursor in das Suchfeld und stellt die ausgewählten Kronkorken dar (`darstellen`).

```
If Not Page.IsPostBack Then

    initialisierung()

End If

TextBox_suche.Focus()

darstellen()
```

Das Befüllen der Auswahllisten darf nur beim ersten Aufruf der Seite erfolgen (`Not Page.IsPostBack`), damit die Auswahl des vorherigen Aufrufs erhalten bleibt.

7.2 initialisierung

Während der Initialisierung lesen wir die Kronkorkendaten aus der Datenbank und befüllen die Auswahllisten mit den Daten.

Dazu definieren wir den Pfad zur XML-Datei mit Hilfe des Befehls `MapPath`, der den relativen Pfad in den korrekten absoluten Pfad im Serverdateisystem übersetzt

```
Dim kronkorken_datei = MapPath("kronkorken.xml")
```

und laden die XML-Datei:

```
Dim kronkorken_xml = XElement.Load(kronkorken_datei)
```

Ab jetzt können wir aus der Sprache heraus direkt auf die Daten der Datenbank zugreifen. Der Befehl

```
Dim alle_getraenke =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    kronkorken.<getraenk>.Value
    Order By getraenk Distinct
```

liefert uns in der Variablen `alle_getraenke` eine alphabetisch sortierte (`Order By`) „Liste" aller in der Datenbank vorhandenen Getränkenamen. Dabei sorgt `Distinct` dafür, dass jeder Getränkename nur einmal auftritt.

Die Liste verwenden wir dann als Datenquelle der Getränke-Auswahlliste

```
DropDownList_getraenk.DataSource = alle_getraenke
```

und stellen die Auswahlliste mit dem `DataBind`-Befehl dar:

```
DropDownList_getraenk.DataBind()
```

Um die Anzahl der zu übertragenden und darzustellenden Kronkorken überschaubar zu halten, wählen wir schon in der anfänglichen Darstellung der Liste auf der Übersichtsseite (Abbildung 2.1) den Eintrag Bier Alkoholfrei aus:

```
DropDownList_getraenk.SelectedValue = "Bier Alkoholfrei"
```

Auf die gleiche Weise füllen wir auch alle übrigen Auswahllisten:

```
Dim alle_marken =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    kronkorken.<marke>.Value
    Order By marke Distinct

DropDownList_marke.DataSource = alle_marken
DropDownList_marke.DataBind()

:

Dim alle_von =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    kronkorken.<von>.Value
    Order By von Distinct

DropDownList_von.DataSource = alle_von
DropDownList_von.DataBind()
```

7.3 darstellen

Im Unterprogramm `darstellen` werten wir die Auswahllisten aus und stellen die ausgewählten Kronkorken als Bilder oder Listen dar.

Dazu lesen wir als erstes die XML-Datei der Kronkorkendaten ein, um später per LINQ auf die Daten zugreifen zu können:

```
Dim kronkorken_xml = XElement.Load(MapPath("kronkorken.xml")
    )
```

Um Groß- und Kleinschreibung bei der Suche nicht zu unterscheiden, wandeln wir den in das Suchfeld eingegebenen Text in Kleinbuchstaben um:

```
Dim suche_lower = TextBox_suche.Text.Trim.ToLower
```

Für die Datenbankabfrage verwenden wir dann sowohl die in den Listen ausgewählten Einträge als auch den Eintrag des Suchfeldes:

```
Dim ausgewaehlte_korken =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    id = kronkorken.<id>.Value,
    :
    tauschen = kronkorken.<tauschen>.Value
    Where (getraenk = DropDownList_getraenk.SelectedValue Or
        DropDownList_getraenk.SelectedValue = "") _
    And (marke = DropDownList_marke.SelectedValue Or
       DropDownList_marke.SelectedValue = "") _
    :
    And (tauschen <> "0" Or DropDownList_tauschen.
       SelectedValue = "") _
    And (getraenk.ToLower.Contains(suche_lower) _
    Or marke.ToLower.Contains(suche_lower) _
    :
    Or id.ToLower.Contains(suche_lower)) _
    And id <> "42"
```

Dabei verbinden wir die einzelnen ausgewählten Einträge der Listen mit einem logischen `Und` (es sei denn, sie sind leer) und vergleichen anschließend der Inhalt des Suchfeldes mit allen (kleingeschriebenen) Datenbankeinträgen. Durch das letzte `And` schließen wir den ersten, leeren Dummy-Kronkorken in der Datenbank aus (Kapitel 6).

Leider haben wir keine einfache Möglichkeit gefunden, im `Order By` Ausdruck einer LINQ-Anweisung dynamisch eine Zeichenkette zu verwenden. Wir verwenden daher nachträglich ein etwas peinliches `If-ElseIf`-Bedingungsmonster für die vom Nutzer vorgegebene Sortierung der Kronkorken:

```
If RadioButtonList_sort.SelectedValue = "farbe" Then

    ausgewaehlte_korken =
        From kronkorken In ausgewaehlte_korken
        Select kronkorken
        Order By CInt(kronkorken.farbe)

:

ElseIf RadioButtonList_sort.SelectedValue = "id" Then

    ausgewaehlte_korken =
        From kronkorken In ausgewaehlte_korken
        Select kronkorken
        Order By kronkorken.id

End If
```

Nach der Ausgabe der Anzahl der gefundenen Kronkorken

```
Label_n_caps.Text = ausgewaehlte_korken.Count
```

unterscheiden wir, ob der Nutzer die Kronkorkenbilder oder (auch) die Liste der Kronkorkendaten sehen möchte. Wenn er nur die Bilder sehen will

```
If RadioButtonList_bildtext.SelectedValue = "bilder" Then
```

machen wir das Bilderfeld sichtbar und verstecken das Listenfeld:

```
Panel_bilder.Visible = True
Panel_beides.Visible = False
```

In Abhängigkeit davon, welche Bildgröße der Nutzer gewählt hat, definieren wir das passende Bildverzeichnis:

```
Dim bild_verzeichnis As String

If RadioButtonList_pixel.SelectedIndex = 0 Then

    bild_verzeichnis = "40x40/"

Else

    bild_verzeichnis = "300x300/"

End If
```

In der anschließenden Schleife über alle ausgewählten Kronkorken

```
For Each korken In ausgewaehlte_korken
```

erzeugen wir für jeden Kronkorken einen Imagebutton

```
Dim bild As New System.Web.UI.WebControls.ImageButton
```

definieren das Unterprogramm, das ausgeführt wird, wenn wir auf ein Kronkorkenbild klicken

```
AddHandler bild.Click, AddressOf bild_Click
```

und verstecken die Kronkorken-Id im alternativen Text des Bildes, da wir sie später im Auswertungsunterprogramm brauchen:

```
bild.AlternateText = korken.id
```

Den Pfadnamen des Kronkorkenbildes basteln wir aus dem passenden Verzeichnisnamen und der Id des Kronkorkens zusammen:

```
bild.ImageUrl = bild_verzeichnis & korken.id & ".jpg"
```

Anschließend füllen wir den Tooltip[1], der erscheint, wenn der Nutzer mit dem Cursor einen Augenblick über dem Kronkorkenbild verweilt, mit den Daten aus der Datenbank

```
bild.ToolTip = _
"Drink: " & korken.getraenk & _
Chr(13) & Chr(10) & _
:
"Id: " & korken.id
```

und fügen das Bild dem Bilderfeld hinzu:

```
Panel_bilder.Controls.Add(bild)

next
```

Wenn der Nutzer die Listenansicht gewählt hat

```
else
```

verstecken wir das Bilderfeld und machen das Listenfeld sichtbar

```
Panel_bilder.Visible = False
Panel_beides.Visible = True
```

Wir erzeugen eine Tabelle[2] mit den entsprechenden Überschriften

[1] Freundlicherweise erlauben mittlerweile alle modernen Browser das Formatieren mehrzeiliger Tooltips. Den Zeilenumbruch realisieren wir dabei mittels „Wagenrücklauf und Zeilenvorschub" (Carriage Return Line Feed: `Chr(13) & Chr(10)`).

[2] Auf diese Weise können wir Überschriften, Spaltenanordnung und -inhalte frei definieren.

```
Dim data_table_korken As New System.Data.DataTable

data_table_korken.Columns.Add("Drink")
:
data_table_korken.Columns.Add("Id")
```

und starten eine Schliefe über alle ausgewählten Kronkorken:

```
For Each korken In ausgewaehlte_korken
```

In der Schleife erzeugen wir (für jeden Kronkorken) eine neue Tabellenzeile

```
Dim data_row_korken = data_table_korken.NewRow
```

tragen in dieser Zeile die Kronkorkendaten ein

```
data_row_korken("Drink") = korken.getraenk
:
data_row_korken("Id") = korken.id
```

und hängen die neue Zeile unten an die Tabelle an:

```
data_table_korken.Rows.Add(data_row_korken)

Next
```

Die fertige Tabelle stellen wir dann in einem `GridView`[3] dar:

```
GridView_korken.DataSource = data_table_korken
GridView_korken.DataBind()
```

Das Erzeugen der anklickbaren Kronkorkenbilder haben wir hier aus Gründen der Übersichtlichkeit in ein extra Unterprogramm ausgelagert:

```
image_buttons_erzeugen()

End If
```

7.4 image_buttons_erzeugen

In einer Schleife über alle Zeilen der Tabelle (und damit über alle Kronkorken)

```
For Each row As GridViewRow In GridView_korken.Rows
```

erzeugen wir wieder Bildschaltflächen

[3] Durch die Trennung von Inhalt (`DataTable`) und Form (`GridView`) sind wir sehr flexibel in der Ausgestaltung und Formatierung unserer Tabelle.

```
Dim bild As New System.Web.UI.WebControls.ImageButton
```

definieren das Unterprogramm, das beim Anklicken aufgerufen wird

```
AddHandler bild.Click, AddressOf bild_Click
```

und verstecken die Kronkorken-Id im Alternativtext des Bildes

```
bild.AlternateText = row.Cells(9).Text
```

Jetzt unterscheiden wir noch (in Abhängigkeit von der Auswahl des Nutzers), ob in der ersten Spalte das echte 40×40 Bild des Kronkorkens dargestellt wird, was bei einer großen Kronkorkenanzahl deutlich länger dauern kann, oder ob für alle Kronkorken nur ein Ersatzbild angezeigt wird:

```
If RadioButtonList_bildtext.SelectedValue = "beides" Then

    bild.ImageUrl = "40x40/" & row.Cells(9).Text & ".jpg"

Else

    bild.ImageUrl = "blauer_knopf.jpg"

End If
```

Schließlich fügen wir die jeweilige Bildschaltfläche in die erste Spalte der Tabelle ein:

```
row.Cells(0).Controls.Add(bild)

Next
```

7.5 bild_Click

Dieses Unterprogramm wird aufgerufen, wenn der Nutzer auf das Bild eines Kronkorkens klickt. Es ruft dann die Detailseite cap.aspx (Kapitel 3) auf, liest die Kronkorken-Id aus dem Alternativtext des aufrufenden Kronkorkens und übergibt die Id als QueryString an die Detailseite:

```
Response.Redirect("cap.aspx?id=" & sender.AlternateText)
```

8 cap.aspx

Die Seite `cap.aspx` stellt ein 300 × 300 Bild und die Daten des Kronkorkens dar und ermöglicht es Nutzern, Korrekturvorschläge per E-Mail zu senden. Der Administrator kann sich auf dieser Seite anmelden und dann alle Daten des aktuellen Kronkorkens in der Datenbank ändern oder den Kronkorken löschen.

8.1 Page_LoadComplete

Als erstes markieren wir Back to the previous page als Defaultschaltfläche, die automatisch ausgeführt wird, wenn der Nutzer die Enter-Taste betätigt:

```
Button_back.Focus()
```

Als nächstes unterscheiden wir, ob der Nutzer als Administrator angemeldet ist und damit das Recht hat, Einträge in der Datenbank zu ändern, oder ob er die Seite nur als „Gast" aufgerufen hat. Wenn er sich schon authentisiert hat

```
If Session("angemeldet") = True Then
```

blenden wir das Kennwortfeld und die Login Schaltfläche aus und die Logout Schaltfläche ein:

```
TextBox_login.Visible = False
Button_login.Visible = False
Button_logout.Visible = True
```

Da nur der Administrator den Rahmen, den Spender und die Anzahl der Tauschkronkorken wissen kann, machen wir diese Informationen nur für den Administrator editierbar und die entsprechenden Auswahllisten sichtbar:

```
TextBox_rahmen.Enabled = True
TextBox_von.Enabled = True
TextBox_tauschen.Enabled = True

DropDownList_rahmen.Visible = True
DropDownList_von.Visible = True
```

Nur der Administrator soll die Möglichkeit haben, Änderungen abzuspeichern, neue Kronkorken einzugeben oder vorhandene Kronkorken zu löschen:

```
Button_speichern.Visible = True
Button_neu.Visible = True
Button_loeschen.Visible = True
```

Als Administrator ergibt das Senden eines Kommentars keinen Sinn:

```
Label_gesendet.Visible = False
TextBox_kommentar.Enabled = False
Button_absenden.Visible = False
```

Wenn der Nutzer kein Administrator ist

```
Else
```

setzen wir alle Eigenschaften auf ihre entgegengesetzten Werte:

```
TextBox_login.Visible = True
:
Button_absenden.Visible = True

End If
```

Das Füllen der Textfelder und Auswahllisten soll nur beim allerersten Aufruf der Seite erfolgen:

```
If Not Page.IsPostBack Then
```

Die Id des aktuellen Kronkorkens entnehmen wir dem QueryString, der beim Aufruf der Seite an die URL angehängt wird (Kapitel 3):

```
Dim korken_id = Request.QueryString("id")
```

Aus der Id erzeugen wir den relativen Pfad zum darzustellenden 300×300 Bild:

```
Image_cap.ImageUrl = "300x300/" & korken_id & ".jpg"
```

Um an die Daten des Kronkorkens heranzukommen, laden wir die XML-Datei

```
Dim kronkorken_datei = MapPath("kronkorken.xml")
Dim kronkorken_xml = XElement.Load(kronkorken_datei)
```

und suchen den Kronkorken mit der aktuellen Id:

```
Dim korken =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    id = kronkorken.<id>.Value,
    :
    tauschen = kronkorken.<tauschen>.Value
    Where id = korken_id
```

Die Daten des Kronkorkens füllen wir dann in die entsprechenden Textfelder:

```
TextBox_getraenk.Text = korken.FirstOrDefault.getraenk
:
TextBox_ip.Text = Request.ServerVariables("remote_addr")
```

Dabei benutzen wir das Auswahlkriterium `FirstOrDefault`, da – zumindest theoretisch – die Möglichkeit bestünde, in der Datenbank mehrere Kronkorken mit der gleichen Id zu finden. Zusätzlich zu den Kronkorkendaten zeigen wir – vielleicht als kleine Motivationshilfe, nur ernstgemeinte Korrekturvorschläge abzuschicken – die IP-Adresse des Nutzers an.

Um für die Korrekturvorschlags-E-Mail zu erkennen, welche Einträge der Nutzer geändert hat, speichern wir die „alten" Einträge im Tooltip des jeweiligen Textfeldes ab:

```
TextBox_getraenk.ToolTip = TextBox_getraenk.Text
:
TextBox_kommentar.ToolTip = TextBox_kommentar.Text
```

Damit auch andere Seiten auf die aktuellen Einträge zugreifen können, speichern wir sie in der anwendungsweiten Session-Variablen ab:

```
Session("getraenk") = TextBox_getraenk.Text.Trim
:
Session("von") = TextBox_von.Text.Trim
```

Das Füllen der Auswahllisten haben wir in ein extra Unterprogramm ausgelagert:

```
dropdownlisten_fuellen()

End If
```

8.2 dropdownlisten_fuellen

Zum Füllen der Auswahllisten laden wir die XML-Datei

```
Dim kronkorken_xml = XElement.Load(MapPath("kronkorken.xml")
   )
```

rufen – wie schon in Abschnitt 7.2 – eindeutige Listen von Getränken, Marken, ... aus der Datenbank ab und binden diese an die entsprechenden Auswahllisten.

```
Dim alle_getraenke =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    kronkorken.<getraenk>.Value
    Order By getraenk
```

```
    Distinct

DropDownList_getraenk.DataSource = alle_getraenke
DropDownList_getraenk.DataBind()
DropDownList_getraenk.SelectedValue = TextBox_getraenk.Text

:

Dim alle_von =
    From kronkorken In kronkorken_xml.<kronkorken>
    Select
    kronkorken.<von>.Value
    Order By von
    Distinct

DropDownList_von.DataSource = alle_von
DropDownList_von.DataBind()
DropDownList_von.SelectedValue = TextBox_von.Text
```

Anschließend wählen wir jeweils in der Auswahlliste schon einmal den Wert vor, der momentan im entsprechenden Textfeld steht.

8.3 Button_absenden_Click

Dieses Unterprogramm wird aufgerufen, wenn der Nutzer die Send correction suggestion Schaltfläche betätigt, um einen Korrekturvorschlag per E-Mail zu senden (Abschnitt 3.1). Wir untersuchen daher im Programm, welche Einträge er geändert hat und basteln daraus die entsprechende E-Mail zusammen.

Anhand der Fallunterscheidungsvariable `wurde_geaendert` erkennen wir am Ende dieses Unterprogrammes, ob wir überhaupt eine E-Mail senden müssen. Als erstes initialisieren wir die Variable daher auf den Fall, dass nichts geändert wurde:

```
Dim wurde_geaendert = False
```

Dann verwenden wir die Kronkorken-Id aus dem Id-Textfeld, um die Links zum aktuellen Kronkorkenbild

```
Dim image_url = "http://suse.thebuchis.de/crown/300x300/" &
   TextBox_id.Text & ".jpg"
```

und seiner Detailseite zu definieren:

```
Dim link_url = "http://suse.thebuchis.de/crown/cap.aspx?id="
    & TextBox_id.Text
```

Aus diesen beiden URLs basteln wir die erste HTML-Zeile der E-Mail (Abbildung 3.2) zusammen, die das Kronkorkenbild anzeigt und beim Anklicken direkt zur Detailseite wechselt:

```
Dim email As String = "<a href='" & link_url & "'><img src='
    " & image_url & "'></a> <br> <br>"
```

Die nachfolgenden Befehle fügen jeweils weitere Zeilen zum E-Mail-Körper hinzu (&=). Dazu lesen wir den alten (potenziell falschen) Eintrag aus dem Tooltip und fügen ihn in die E-Mail ein:

```
email &= _
"Getränk: " & _
TextBox_getraenk.ToolTip
```

Wenn der Nutzer den Eintrag des aktuellen Textfeldes geändert hat

```
If Not TextBox_getraenk.Text = TextBox_getraenk.ToolTip Then
```

merken wir uns, dass eine Änderung stattgefunden hat und die E-Mail gesendet werden muss

```
wurde_geaendert = True
```

und fügen den Änderungsvorschlag fettgedruckt nach einem „Pfeil" in die gleiche Zeile ein:

```
email &= _
" -----> " & _
"<b>" & _
TextBox_getraenk.Text & _
"</b>"

End If
```

Auf die gleiche Weise untersuchen wir dann auch die übrigen vom Nutzer veränderbaren Felder auf Änderungen hin und tragen sie gegebenenfalls in die E-Mail ein.

Wenn wir alle Änderungsmöglichkeiten analysiert haben, untersuchen wir, ob der Nutzer überhaupt irgendeine Änderung vorgeschlagen (oder einen Kommentar abgegeben) hat:

```
If wurde_geaendert Then
```

Nur in diesem Fall soll die E-Mail tatsächlich gesendet werden und wir fügen noch die vom Nutzer nicht veränderbaren Daten (Id, Rahmen und IP-Adresse) hinzu:

```
email &= _
"<br>" & _
"ID: " & _
TextBox_id.Text & _
```

```
"<br>" & _
"Rahmen: " & _
TextBox_rahmen.Text & _
"<br>" & _
"IP: " & _
TextBox_ip.Text
```

Um die E-Mail versenden zu können, müssen wir den Absender

```
Dim absender As New System.Net.Mail.MailAddress("
   suse@thebuchis.de")
```

und den Empfänger definieren

```
Dim empfaenger As New System.Net.Mail.MailAddress("susanne.
   buchholz@gmail.com")
```

und daraus ein E-Mail-Objekt erzeugen:

```
Dim Nachricht As New System.Net.Mail.MailMessage(absender,
   empfaenger)
```

Wir definieren die Betreff-Zeile

```
Nachricht.Subject = "Kronkorkenkorrektur"
```

legen fest, dass es sich um eine HTML-E-Mail handelt, damit wir das Kronkorkenbild einbinden und die Formatierungen verwenden können

```
Nachricht.IsBodyHtml = True
```

und verwenden die vorher zusammengebastelte Zeichenkette als E-Mail-Inhalt:

```
Nachricht.Body = email
```

Jetzt müssen wir nur noch den Mailserver definieren

```
Dim Client As New System.Net.Mail.SmtpClient

Client.Host = "fbm-server.fbm.hs-bremen.de"
```

und können die E-Mail absenden:

```
Client.Send(Nachricht)
```

Wenn wir eine E-Mail gesendet haben, informieren wir den Nutzer darüber, indem wir die Zeichenkette Your correction suggestion has been sent. sichtbar machen

```
Label_gesendet.Visible = True
```

und die Schaltfläche gegen erneutes Senden sperren:

```
Button_absenden.Enabled = False

End If
```

8.4 Button_login_Click

Wenn der Administrator das Kennwort in das dafür vorgesehene Textfeld eingetragen hat und auf die Login Schaltfläche klickt, berechnen wir den SHA1-Wert des Kennwortes

```
Dim kennwort_hash = FormsAuthentication.
   HashPasswordForStoringInConfigFile(TextBox_login.Text, "
   SHA1")
```

und vergleichen ihn mit dem vorher offline berechneten Hashwert des Kennwortes. Wenn die beiden identisch sind

```
If kennwort_hash = "12345...67890" Then
```

setzen wir die Sessionvariable `angemeldet` auf wahr.

```
Session("angemeldet") = True

End If
```

Auf anderen Seiten dieses Programms können wir dann die Sessionvariable auslesen, feststellen, ob der Administrator angemeldet ist und gegebenenfalls die Funktion der Seiten anpassen.[1]

8.5 Button_logout_Click

Ein Klick auf die Logout Schaltfläche setzt die entsprechende Sessionvariable auf falsch und meldet damit den Administrator ab:

```
Session("angemeldet") = False
```

8.6 Button_loeschen_Click

Um einen Kronkorken komplett aus dem Datenbestand zu entfernen, müssen wir ihn nicht nur aus der Datenbank entfernen, sondern wir sollten auch seine dann ja verwaisten Bilder von der Platte putzen.

Wenn der Administrator also die – ganz bewusst rechts oben positionierte – Schaltfläche Diesen Kronkorken ohne Sicherheitsabfrage löschen betätigt, wird die Id des aktuellen Kronkorken aus dem QueryString gelesen

[1] Natürlich ist uns klar, dass SHA1-Hashes mit ein wenig krimineller Energie geknackt werden können und dass das Setzen einer Sessionvariablen keinen sicheren Anmeldeschutz bietet. Wir bitten daher potenzielle Hacker einfach darum, dieses wenig lukrative Programm in Frieden arbeiten zu lassen und ihre Energie lieber in finanziell attraktivere Projekte zu investieren.

```
Dim korken_id = Request.QueryString("id")
```

und die Verbindung zur XML-Datei hergestellt:

```
Dim kronkorken_datei = MapPath("kronkorken.xml")
Dim kronkorken_xml = XElement.Load(kronkorken_datei)
```

Mit dem passenden LINQ-Befehl sprechen wir genau den aktuellen Kronkorken an

```
Dim korken =
    From kronkorken In kronkorken_xml.<kronkorken>
    Where kronkorken.<id>.Value = korken_id
```

und entfernen ihn aus dem Datenbestand des laufenden Programms:

```
korken.Remove()
```

Das Zurückschreiben der XML-Datei löscht ihn dann auch dauerhaft auf der Platte:

```
kronkorken_xml.Save(kronkorken_datei)
```

Jetzt müssen wir nur noch die Bilder des Kronkorkens löschen. Dazu basteln wir den absoluten Pfad des 300×300 Bildes mit Hilfe des `MapPath`-Befehls zusammen:

```
Dim korken_300_datei_name = MapPath( _
    "300x300/" & _
    korken_id & _
    ".jpg")
```

und löschen das Bild, wenn es noch existiert:[2]

```
If IO.File.Exists(korken_300_datei_name) Then

    IO.File.Delete(korken_300_datei_name)

End If
```

Das Löschen des 40×40 Bildes geschieht analog zum Löschen des 300×300 Bildes.

Nach dem kompletten Löschen des aktuellen Kronkorkens ist der Versuch, weiterhin „seine” Detailseite anzuzeigen, natürlich sinnlos und würde sogar zu einer Fehlermeldung des Programms führen. Wir wechseln daher zurück zur Übersichtsseite:

```
Response.Redirect("default.aspx")
```

[2] Normalerweise wird das Bild natürlich existieren. Die If-Bedingung ist daher eine reine Angstabfrage, um zu verhindern, dass das Programm schon vor dem Löschen des 40×40 Bildes abbricht, wenn – aus welchen Gründen auch immer – das 300×300 Bild doch nicht (mehr) vorhanden ist.

8.7 Button_speichern_Click

Genau wie beim Löschen eines Kronkorkens greifen wir auch zum Abspeichern der Änderungen auf der Detailseite direkt auf den aktuellen Kronkorken in der XML-Datei zu:

```
Dim korken_id = Request.QueryString("id")

Dim kronkorken_datei = MapPath("kronkorken.xml")
Dim kronkorken_xml = XElement.Load(kronkorken_datei)

Dim korken =
    From kronkorken In kronkorken_xml.<kronkorken>
    Where kronkorken.<id>.Value = korken_id
```

Wir aktualisieren alle seine Daten aus den entsprechenden Textfeldern

```
korken.<getraenk>.Value = TextBox_getraenk.Text.Trim
:
korken.<tauschen>.Value = TextBox_tauschen.Text.Trim
```

und speichern die Änderungen ab:

```
kronkorken_xml.Save(kronkorken_datei)
```

Zum Schluss informieren wir noch den Nutzer mit dem Satz: Deine Änderungen wurden abgespeichert.

```
Label_abgespeichert.Visible = True
```

8.8 DropDownList_getraenk_SelectedIndexChanged

Die Auswahllisten in der rechten Spalte der Detailseite besitzen die Eigenschaft AutoPostBack; Das Auswählen eines neuen Eintrags sendet die Seite daher direkt zurück an den Server, so dass wir den neuen Eintrag sofort in das passende Textfeld übertragen können:

```
TextBox_getraenk.Text = DropDownList_getraenk.SelectedValue
```

Die übrigen Auswahllisten reagieren entsprechend.

8.9 Button_neu_Click

Die für den Administrator sichtbare Schaltfläche Zurück zur Auswahl ruft direkt die Seite auf, auf der wir einen neuen Kandidaten zur Aufnahme in die Datenbank bestimmen:

```
Response.Redirect("kronkorken_auswaehlen.aspx")
```

9 kronkorken_auswaehlen.aspx

Auf der Seite `kronkorken_auswaehlen.aspx` können wir einen Scan von vielen Kronkorken auf den Server hochladen und aus dem Scan einen einzelnen Kronkorken auswählen. Die Seite besteht aus einem server- und einem clientbasierten Teil. Die Abschnitte 9.1 und 9.2 beschreiben dabei die serverseitigen ASP.NET-Unterprogramme, während die Abschnitte 9.3 bis 9.5 die clientseitigen JavaScript-Programme erläutern.

9.1 Page_LoadComplete

Abhängig davon, ob der Administrator angemeldet ist, bieten wir dem Nutzer auf der Seite unterschiedliche Kontrollelemente an. Im Administrator-Modus (Abbildung 4.2b)

```
If Session("angemeldet") = True Then
```

machen wir die Schaltflächen Datei auswählen, Hochladen und Logout sichtbar und verstecken die Login Schaltfläche und ihr Textfeld:

```
FileUpload1.Visible = True
Button_hochladen.Visible = True
TextBox_login.Visible = False
Button_login.Visible = False
Button_logout.Visible = True
```

Der normale Nutzer (Abbildung 4.2a)

```
Else
```

bekommt nur die Möglichkeit, sich anzumelden:

```
FileUpload1.Visible = False
Button_hochladen.Visible = False
TextBox_login.Visible = True
Button_login.Visible = True
Button_logout.Visible = False

End If
```

Die beiden Unterprogramme `Button_login_Click` und `Button_logout_Click` sind identisch mit den in Abschnitt 8.4 und Abschnitt 8.5 beschriebenen Unterprogrammen.

9.2 Button_hochladen_Click

Um einen neuen Scan auf den Server hoch zu laden, klickt der Nutzer auf Datei auswählen, wählt den Scan im eigenen Dateisystem aus und drückt dann Hochladen.

Als Dateinamen auf dem Server haben wir fest den Namen `scan.jpg` vorgegeben:

```
Dim dateiname_auf_server As String = "E:\inetpub\suse.
   thebuchis.de\crown\scan.jpg"
```

Wenn die Auswahl der Scandatei auf dem eigenen Rechner erfolgreich war

```
If (FileUpload1.HasFile) Then
```

laden wir den Scan auf den Server hoch:

```
FileUpload1.SaveAs(dateiname_auf_server)

End If
```

Um die in 4.13 dargestellten weißen Quadrate für den neuen Scan zu entfernen, löschen wir außerdem durch Drücken auf Hochladen mit einem Inline-JavaScript-Befehl den lokalen Speicher, in dem die Positionen der Quadrate gehalten werden:

```
<asp:Button
    ID="Button_hochladen"
    runat="server"
    Text="Hochladen"
    Visible="False"
    OnClientClick="return localStorage.clear()" />
```

9.3 <script>

HTML5 bietet uns mit dem (neuen) `canvas`-Element

```
<canvas
    id="leinwand"
    width="540"
    height="750"></canvas>
```

sehr elegante Möglichkeiten der interaktiven Bildbearbeitung. Mit der `getContext("2d")`-Methode

```
var kontext = leinwand.getContext("2d")
```

erhalten wir einen Kontext, auf dem wir den Scan darstellen und bearbeiten können. Dazu laden wir den Scan in ein neues Bild

```
var bild = new Image()
bild.src = "scan.jpg"
```

und stellen ihn im folgenden Unterprogramm dar.

9.4 bild.onload

Durch die Verwendung der `bild.onload`-Methode warten wir mit unseren eigenen Bearbeitungsaktionen, bis der Browser den Scan komplett geladen hat:

```
bild.onload = function () {
```

Jetzt können wir den Scan auf dem Kontext darstellen:

```
kontext.drawImage(bild, 0, 0, bild.width / 5, bild.height / 5)
```

Wenn unser Scanner etwa eine DIN A4-Fläche (210 mm × 279 mm) einscannt, entspricht dies bei einer Auflösung von 300 dpi und ein paar zusätzlichen Randpunkten einem exemplarischen Bildformat von 2550 × 3501 Bildpunkten. Wir komprimieren den Scan daher um den Faktor 5, damit er komplett in das Bildformat der Leinwand (540 × 750 Bildpunkte) hinein passt.

Als nächstes zeichnen wir die weißen Quadrate über die schon bearbeiteten Kronkorken. Dazu laden wir die Liste der Punkte, die der Nutzer vorher angeklickt hatte, aus dem lokalen Speicher:[1]

```
local_storage_stempel = localStorage.stempel
```

Wenn der Nutzer vorher schon Kronkorken verarbeitet hatte[2]

```
if (local_storage_stempel != undefined) {
```

wandeln wir die angeklickten Punkte aus dem zum Abspeichern von Listen notwendigen JSON-Format in ein einfaches Feld:

```
stempel = JSON.parse(local_storage_stempel)
```

Jetzt können wir eine Schleife über alle Punkte laufen lassen:

```
for (i = 0; i < stempel.length; i += 2) {
```

In der Schleife zeichnen wir jeweils ein Quadrat. Dazu beginnen wir ein neues Zeichenelement

```
kontext.beginPath()
```

[1] Das in HTML5 neu hinzugekommene `localStorage`-Objekt erlaubt es uns, Informationen selbst über einen Neustart des Browsers hinweg aufzubewahren.

[2] Diese Abfrage ist notwendig, da das in der nächsten Zeile durchgeführte Parsen bei einem nicht definierten Objekt zu einem Fehler führen würde.

zeichnen ein Quadrat der Kantenlänge 20 Bildpunkte und definieren seinen Mittelpunkt an der Stelle, an der der Nutzer vorher den Kronkorken ausgewählt hatte:

```
kontext.rect(stempel[i] - 10, stempel[i + 1] - 10, 20, 20)
```

Schließlich füllen wir das Quadrat weiß (ohne Umrandung)

```
kontext.fillStyle = 'white'
```

und stellen es dar:

```
kontext.fill()
}}}
```

9.5 leinwand.onclick

Das Unterprogramm `leinwand.onclick` wird aufgerufen, wenn der Nutzer irgendwo auf den Scan klickt

```
leinwand.onclick = function (event) {
```

Wir möchten dann die Koordinaten des Klicks bezogen auf die linke obere Ecke des Scans ermitteln. Peinlicherweiser herrscht allerdings selbst bei modernen Browsern nach wie vor ein heilloses Durcheinander, was das Abfragen von relativen Klickkoordinaten anbelangt. Wenn der Browser die Methoden `offsetX` und `offsetY` kennt, benutzen wir diese. Wenn nicht, verwenden wir die Methoden `layerX` und `layerY`, die die Klickkoordinaten bezogen auf die linke obere Ecke des nächst höheren Objekts liefern und ziehen dessen Koordinaten ab:

```
var offset_x = (event.offsetX !== undefined) ? event.offsetX : (
   event.layerX - event.target.offsetLeft)

var offset_y = (event.offsetY !== undefined) ? event.offsetY : (
   event.layerY - event.target.offsetTop)
```

Für die weitere Verwendung auf anderen Seiten speichern wir die Rohdaten der Klickkoordinaten in der Originalauflösung des Scans im lokalen Speicher ab:

```
localStorage.x_roh = offset_x * 5
localStorage.y_roh = offset_y * 5
```

Um unterscheiden können, ob schon andere Kronkorken angeklickt wurden, laden wir das entsprechende Objekt probehalber aus dem lokalen Speicher:

```
local_storage_stempel = localStorage.stempel
```

Wenn der Nutzer vorher noch keine anderen Kronkorken angeklickt hatte

```
if (local_storage_stempel == undefined) {
```

definieren wir eine neue Liste

```
var stempel = new Array()
```

Wenn es schon vorhandene Koordinaten gibt,

```
} else {
```

laden wir diese aus dem lokalen Speicher und wandeln sie aus dem zeichenkettenbasierten JSON-Format in eine „normale" Liste:

```
stempel = JSON.parse(local_storage_stempel)
}
```

Die aktuellen Klickkoordinaten hängen wir ans Ende der Koordinatenliste an (erst die x-Koordinate, dann die y-Koordinate)

```
stempel.push(offset_x)
stempel.push(offset_y)
```

wandeln die Liste in das zeichenkettenorientierte JSON-Format und sichern sie im lokalen Speicher:

```
localStorage.stempel = JSON.stringify(stempel)
```

Schließlich wechseln wir zu nächsten Seite, um dort die Feinjustierung des Ausschnitts und die Drehung des Kronkorkens vorzunehmen:

```
location.href = "ausschnitt_bestimmen.aspx"
}
```

10 ausschnitt_bestimmen.aspx

Auf der Seite `ausschnitt_bestimmen.aspx` hat der Nutzer die Möglichkeit, den Mittelpunkt des Kronkorkens festzulegen und ihn horizontal auszurichten. Wir legen dazu zwei Zeichenflächen an:

```
<canvas
    id="leinwand_800"
    width="800"
    height="800"></canvas>

<canvas
    id="leinwand_300"
    width="300"
    height="300"></canvas>
```

Auf `leinwand_800` führt der Nutzer seine Aktionen durch; `leinwand_300` bleibt unsichtbar und wird am Ende an den Server übertragen.

10.1 <script>

Im „Hauptprogramm" der Seite initialisieren wir die Listen der aktuellen Klickkoordinaten

```
var x = []
var y = []
```

und die Variable, die angibt, ob ein Gitter über den aktuellen Kronkorken gezeichnet werden soll

```
var gitter_an = false
```

und verstecken vorerst die Schaltfläche, die uns später auf die nächste Seite führt:

```
button_senden.hidden = true
```

Wir lesen die Rohdaten des Klicks auf der Seite `kronkorken_auswaehlen` aus dem lokalen Speicher und interpretieren sie als Fließkommazahlen:

```
var x_roh = parseFloat(localStorage.x_roh)
var y_roh = parseFloat(localStorage.y_roh)
```

Der getContext-Befehl liefert uns für beide Zeichenflächen einen Kontext zurück, auf dem wir grafisch arbeiten können:

```
var kontext_800 = leinwand_800.getContext("2d")
kontext_800.strokeStyle = "white"
kontext_800.fillStyle = "white"

var kontext_300 = leinwand_300.getContext("2d")
kontext_300.fillStyle = "white"
```

Flächen sollen dabei weiß gefüllt werden und die Linienfarbe auf der großen Zeichenfläche soll ebenfalls weiß sein.

Des Weiteren definieren wir einen Tooltip, der dem Nutzer die Verwendung der Seite erläutert:

```
leinwand_800.title =
    "Drei Klicks, um den Ausschnitt zu bestimmen \n" +
    "Zwei Klicks, um den Horizont zu bestimmen \n\n" +
    "Dann: \n" +
    "w: Kronkorken nach oben verschieben \n" +
    "a: Kronkorken nach links verschieben \n" +
    "s: Kronkorken nach unten verschieben \n" +
    "d: Kronkorken nach rechts verschieben \n" +
    "q: Kronkorken links herum drehen \n" +
    "e: Kronkorken rechts herum drehen \n\n" +
    "g: Gitter einschalten"
```

Abschließend laden wir den Scan in ein neues Bildobjekt:

```
var bild = new Image()

bild.src = "scan.jpg"
```

10.2 bild.onload

Nachdem der Browser das Bild geladen hat

```
bild.onload = function () {
```

stellen wir einen 400×400 Bildpunkte großen Ausschnitt auf der großen Leinwand dar:

```
kontext_800.drawImage(bild, x_roh - 200, y_roh - 200, 400, 400,
   0, 0, 800, 800)
```

Dabei bauen wir den Ausschnitt symmetrisch um den Klickpunkt (x_roh, y_roh) der vorherigen Seite herum auf (x_roh - 200, y_roh - 200). Um einzelne Details im Folgenden präziser anklicken zu können, verdoppeln wir das dargestellte Bildformat auf

800 × 800.[1] Durch die Verwendung eines 400 × 400-Ausschnitts erreichen wir, dass der 300 × 300 Bildpunkte große Kronkorken auch dann noch vollständig dargestellt wird, wenn der Nutzer nicht genau seine Mitte getroffen hat (Abbildung 10.1).

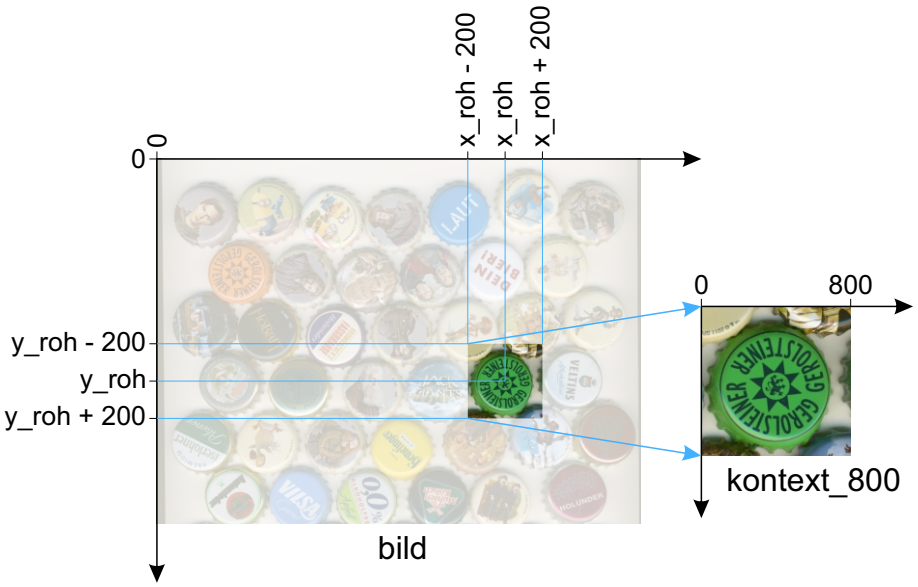

Abbildung 10.1: Symmetrische Ausschnittsvergrößerung um den Klickpunkt herum

Nach jedem erneuten Laden des Ausschnitts setzen wir die Variable `detail_status`, die uns die Anzahl der schon ausgeführten Klicks angibt, wieder zurück:

```
detail_status = 0
}
```

10.3 leinwand_800.onclick

Das Unterprogramm wird jedes Mal ausgeführt, wenn der Nutzer irgendwo auf die große Zeichenfläche klickt:

```
leinwand_800.onclick = function (event) {
```

Bei jedem Klick erhöhen wir den Klickzähler, damit wir nach dem dritten Klick den Ausschnitt und nach dem fünften Klick den Horizont berechnen können:

```
detail_status += 1
```

Wie schon in Abschnitt 9.5 ermitteln wir browserabhängig die Koordinaten des aktuellen Klicks

[1] Anders als in praktisch allen deutschen und amerikanischen Serien sehen wir durch dieses „Hineinzoomen" natürlich nicht mehr Details; gleichwohl wird uns das manuelle „Treffen" markanter Punkte etwas erleichtert.

```
var offset_x = (event.offsetX !== undefined) ? event.offsetX : (
   event.layerX - event.target.offsetLeft)

var offset_y = (event.offsetY !== undefined) ? event.offsetY : (
   event.layerY - event.target.offsetTop)
```

und verwenden den Klickzähler als Index, um sie in den Feldern `x` und `y` abzuspeichern:

```
x[detail_status] = offset_x
y[detail_status] = offset_y
```

10.3.1 Kreuz zeichnen

Die nächsten Zeilen zeichnen ein (weißes) Kreuz an die gerade angeklickte Stelle:

```
kontext_800.beginPath()
kontext_800.moveTo(offset_x - 10, offset_y - 10)
kontext_800.lineTo(offset_x + 10, offset_y + 10)
kontext_800.moveTo(offset_x - 10, offset_y + 10)
kontext_800.lineTo(offset_x + 10, offset_y - 10)
kontext_800.stroke()
```

Dazu bewegen (`moveTo`) wir den Zeichenstift an die linke obere Ecke eines 20×20 Bildpunkte großen Quadrats, das symmetrisch um den aktuellen Klickpunkt liegt (Abbildung 10.2).

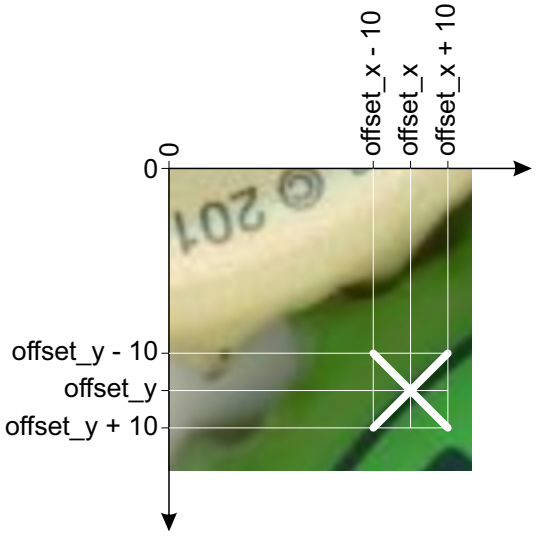

Abbildung 10.2: Weißes Kreuz zeichnen

Mit dem anschließenden `lineTo`-Befehl ziehen wir die erste weiße Linie in die rechte untere Ecke des Quadrats. Das nächste `moveTo` hebt den Zeichenstift und bewegt ihn in

die linke untere Ecke; Das zugehörige `lineTo` erzeugt die zweite Linie zur rechten obere Ecke. Abschließend verwenden wir noch den `stroke`-Befehl, um die Linien tatsächlich sichtbar zu machen.

10.3.2 Mittelpunkt berechnen

Nach dem dritten Klick

```
if (detail_status == 3) {
```

berechnen wir den Kreis, der genau durch die drei Punkte verläuft. Dazu verwenden wir die Formeln, die wir im Anhang A mit Hilfe eines Matlab-Programms hergeleitet haben.[2]

Damit berechnen wir sowohl die x-Koordinate des Mittelpunktes

```
x_m =
    1 / 2 * (
    -y[1] * x[2] * x[2] +
    y[3] * x[2] * x[2] -
    y[2] * y[3] * y[3] +
    y[1] * x[3] * x[3] +
    y[2] * y[2] * y[3] -
    y[2] * y[2] * y[1] +
    y[3] * y[3] * y[1] +
    x[1] * x[1] * y[2] -
    y[3] * y[1] * y[1] +
    y[2] * y[1] * y[1] -
    x[3] * x[3] * y[2] -
    x[1] * x[1] * y[3]
    ) / (
    x[1] * y[2] -
    x[1] * y[3] -
    y[1] * x[2] +
    y[1] * x[3] -
    x[3] * y[2] +
    y[3] * x[2])
```

als auch seine y-Koordinate

```
y_m =
    -1 / 2 * (
    x[1] * x[1] * x[2] -
    x[1] * x[1] * x[3] -
    x[1] * x[2] * x[2] -
```

[2] In der praktischen Ausführung haben wir die Quadrate als Produkte ausgedrückt und die Reihenfolge der Summanden angepasst.

```
    x[1] * y[2] * y[2] +
    x[1] * x[3] * x[3] +
    x[1] * y[3] * y[3] +
    y[1] * y[1] * x[2] -
    y[1] * y[1] * x[3] -
    x[3] * x[3] * x[2] +
    x[3] * x[2] * x[2] +
    x[3] * y[2] * y[2] -
    y[3] * y[3] * x[2]
    ) / (
    x[1] * y[2] -
    x[1] * y[3] -
    y[1] * x[2] +
    y[1] * x[3] -
    x[3] * y[2] +
    y[3] * x[2])
```

aus den x- und y-Koordinaten der drei bislang angeklickten Punkte.

10.3.3 Freistellen

Im nächsten Schritt wollen wir den Kronkorken freistellen. Wir erreichen dies, indem wir über den Kronkorken eine weiße Maske legen, die aus einem Quadrat der Seitenlänge 800 besteht

```
kontext_800.beginPath()
kontext_800.rect(0, 0, 800, 800)
```

aus dem um den gerade berechneten Mittelpunkt ein Kreis mit einem Radius von 300 Bildpunkten ausgeschnitten ist:

```
kontext_800.arc(x_m, y_m, 300, 0, 2 * Math.PI, true)
kontext_800.fill()
}
```

Auf weißem Hintergrund sieht dies dann so aus, als ob das Bild nur noch aus dem freigestellten Kronkorken besteht (Abbildung 4.4).

10.3.4 Drehwinkel berechnen

Die nächsten beiden Klicks definieren den in Abschnitt 4.3.2 beschriebenen und in Abbildung 10.3 dargestellten Drehwinkel Φ, um den das Bild gedreht werden muss, damit die Schrift auf dem Kronkorken waagerecht ausgerichtet ist.

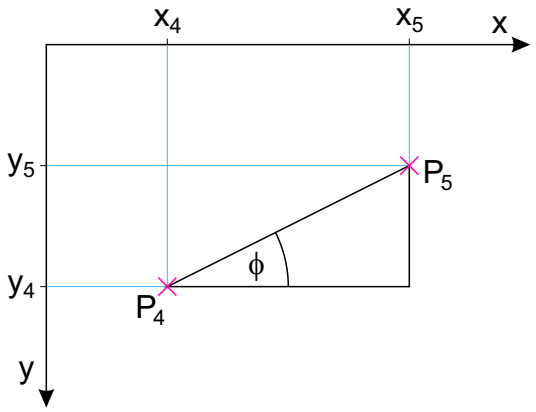

Abbildung 10.3: Drehwinkel aus zwei Punkten berechnen

Dazu berechnen wir im rechtwinkligen Dreieck in Abbildung 10.3 den Tangens des gesuchten Winkels als seine Gegenkathete[3] dividiert durch seine Ankathete:

$$\tan \Phi = \frac{y_4 - y_5}{x_5 - x_4}$$

Den Winkel selbst erhalten wir dann durch die Umkehrfunktion des Tangens:

$$\Phi = \arctan\left(\frac{y_4 - y_5}{x_5 - x_4}\right) \tag{10.1}$$

Nach dem fünften Klick

```
else if (detail_status == 5) {
```

berechnen wir also den gesuchten Drehwinkel mit der `atan2`-Funktion:

```
phi = Math.atan2(y[4] - y[5], x[5] - x[4])
```

Die `atan2`-Funktion hat gegenüber der einfachen Tangensfunktion den Vorteil, dass sie den Zähler und den Nenner von Gleichung (10.1) als zwei einzelne Parameter erhält und daher auch Winkel größer als 90° und kleiner als −90° berechnen kann.

10.3.5 Darstellen

Wir können jetzt die Schaltfläche zum Absenden des Kronkorkens an den Server sichtbar machen

```
button_senden.hidden = false
```

[3] Achtung: Da die y-Achse des Bildkoordinatensystems nach unten positiv ausgerichtet ist, ist in Abbildung 10.3 der y-Wert des Punktes P_4 größer als der des Punktes P_5. Damit die Gegenkathete und damit der Winkel Φ positiv werden, muss also $y_4 - y_5$ gebildet werden.

vorauswählen

```
button_senden.autofocus = true
```

und den ausgeschnittenen und gedrehten Kronkorken darstellen:

```
zeichnen()
}}
```

10.4 zeichnen

Das zeichnen-Unterprogramm

```
function zeichnen() {
```

wird jedes Mal aufgerufen, wenn der Nutzer eine der aktiven Tasten (w, a, s, d, q, e, g) gedrückt hat (Abschnitt 10.7). Im Unterprogramm stellen wir den durch die Tastendrücke verschobenen und gedrehten Ausschnitt (kontext_800) sichtbar dar. Außerdem zeichnen wir den für die Dublettenerkennung genutzten Ausschnitt in Originalgröße unsichtbar auf einem eigenen Kontext (kontext_300).

10.4.1 Drehen und zeichnen

Die Vorgehensweise beim Drehen eines Bildausschnitts unter HTML5 ist etwas gewöhnungsbedürftig [3]. Da die Drehung grundsätzlich um den Ursprung des momentanen Koordinatensystems stattfindet, müssen wir das Koordinatensystem zuerst in den gewünschten Drehpunkt verschieben und dann dort die Drehung durchführen. Da Drehungen nur auf Zeichenobjekte wirken, die nach der Drehung gezeichnet werden, müssen wir den Bildausschnitt dann nochmals zeichnen.

Durch die Befehle save und restore können wir das Koordinatensystem nach der Drehung wieder in seinen ursprünglichen Zustand zurück versetzen, damit spätere Zeichenbefehle wieder im ungedrehten Koordinatensystem durchgeführt werden. Vor der Drehung speichern wir also den Originalzustand des Koordinatensystems ab:

```
kontext_800.save()
```

In diesem unverschobenen und ungedrehten Zustand liegt der Ursprung des Koordinatensystems in der linken oberen Ecke des Bildausschnitts (Abbildung 10.4).

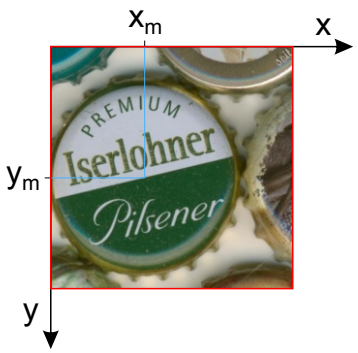

Abbildung 10.4: Unverschoben und ungedreht

Da wir den Mittelpunkt (`x_m`, `y_m`) des Kronkorkens durch drei Klicks schon berechnet haben (Anhang A), können wir jetzt den Ursprung des Koordinatensystems in den Mittelpunkt verschieben:

```
kontext_800.translate(x_m, y_m)
```

Wie in Abbildung 10.5 deutlich wird, würde nach der Verschiebung nur noch der linke obere Anteil des Bildausschnitts rechts unten im rot umrandeten Darstellungsbereich zu sehen sein.

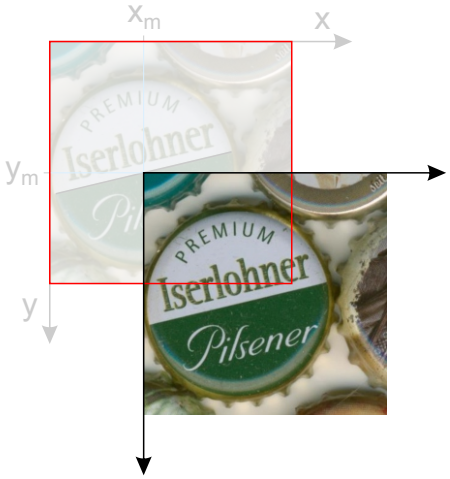

Abbildung 10.5: Verschoben

Auch den Winkel `phi`, mit dem der Bildausschnitt gedreht werden soll, hat der Nutzer vorher durch zwei Klicks definiert. Wir drehen also nun in Abbildung 10.6 den verschobenen Bildausschnitt um seinen (in den Mittelpunkt des Kronkorkens verschobenen) Koordinatenursprung:

```
kontext_800.rotate(phi)
```

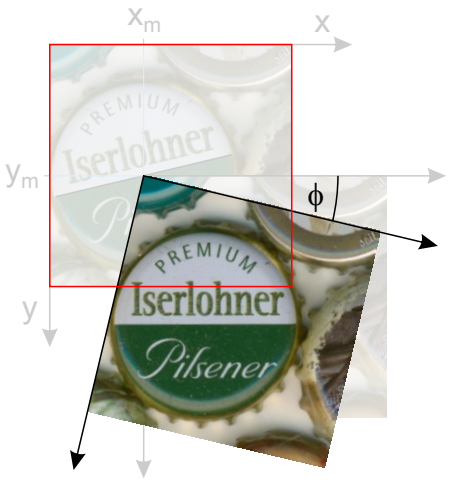

Abbildung 10.6: Verschoben und gedreht

Auch nach der Drehung würde nur der (gedrehte) linke obere Bildanteil im roten Darstellungsbereich sichtbar sein (Abbildung 10.6). Um dies zu verhindern, müssen wir in Abbildung 10.7 den Bildausschnitt im **gedrehten** Koordinatensystem nach links oben verschoben erneut zeichnen:

```
kontext_800.drawImage(bild, x_roh - 200, y_roh - 200, 400, 400,
   -x_m, -y_m, 800, 800)
```

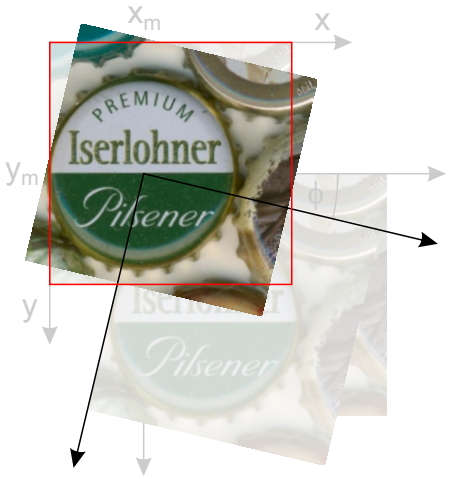

Abbildung 10.7: Verschoben, gedreht und zurück verschoben gezeichnet

Damit dabei der Mittelpunkt des gedrehten Bildausschnitts genau auf dem Mittelpunkt des Originalbildausschnitts (also dem Ursprung des verschobenen Koordinatensystems) landet, müssen wir der linken oberen Ecke des Bildausschnitts im gedrehten Koordinatensystem genau die negativen Koordinaten des Mittelpunkts geben (..., `-x_m, -y_m,` ...).

Wie wir in Abbildung 10.7 sehen, liegt der Kronkorken dann korrekt ausgerichtet an der richtigen Stelle. Allerdings befinden sich die Ecken des Bildausschnitts außerhalb des roten Rahmens und die Ecken des Rahmens selbst bleiben leer. Diese beiden – bei einer Drehung natürlich unvermeidlichen – Probleme haben aber keine Auswirkungen, da wir als nächstes ja wieder eine Maske auf den Bildausschnitt legen, die nur den Kronkorken selbst „ausschneidet".

10.4.2 Freistellen

Dazu verschieben wir für die nachfolgenden Zeichenbefehle den Ursprung des Koordinatensystems wieder zurück in die linke obere Ecke des roten Darstellungsbereichs (Abbildung 10.4)

```
kontext_800.restore()
```

und zeichnen – wie in Abschnitt 10.3.3 – im Originalkoordinatensystem wieder das weiße Rechteck mit dem Kreis als Ausschnitt:

```
kontext_800.beginPath()
kontext_800.rect(0, 0, 800, 800)
kontext_800.arc(x_m, y_m, 300, 0, 2 * Math.PI, true)
kontext_800.fill()
```

10.4.3 Gitter einschalten

Wenn der Nutzer die Taste `g` gedrückt hat, wird in Abschnitt 10.7 die Variable `gitter_an` auf den Wert `true` gesetzt und das in Abschnitt 10.5 beschriebene Unterprogramm `gitter_erzeugen` aufgerufen:

```
if (gitter_an) {
    gitter_erzeugen()
}
```

10.4.4 Unsichtbaren Kronkorken drehen und zeichnen

Für die Dublettensuche und die Datenbank benötigen wir einen 300×300 Bildpunkte großen Ausschnitt exakt um den Mittelpunkt des Kronkorkens, der dann nur den Kronkorken zeigt. Diesen Ausschnitt zeichnen wir auf den in Abschnitt 10.1 erzeugten, unsichtbaren `kontext_300`. Die Vorgehensweise ist dabei genau wie in Abschnitt 10.4.1 beschrieben:

```
kontext_300.save()

kontext_300.translate(150, 150)
```

```
kontext_300.rotate(phi)

kontext_300.drawImage(bild, x_roh - 200 + x_m / 2 - 150, y_roh -
    200 + y_m / 2 - 150, 300, 300, -150, -150, 300, 300)

kontext_300.restore()

kontext_300.beginPath()
kontext_300.rect(0, 0, 300, 300)
kontext_300.arc(150, 150, 150, 0, 2 * Math.PI, true)
kontext_300.fill()
```

Wir speichern den Originalzustand (`save`), verschieben das Koordinatensystem nach unten und rechts in die Mitte des Bildausschnitts (`translate`), rotieren den Ausschnitt mit dem aktuellen Winkel (`rotate`), zeichnen den passenden Bildausschnitt (`drawImage`), verschieben das Koordinatensystem in seinen Originalzustand zurück (`restore`) und stellen den Kronkorken frei (`beginPath` ... `fill`).

Ein bisschen Überlegung erfordert dabei die Definition der Koordinaten des Bildausschnitts im `drawImage`-Befehls. Wir wollen den Bildausschnitt ja jetzt nicht mehr wie in Abschnitt 10.4.1 um den Klick im Scan herum aufbauen (weißes Kreuz in Abbildung 10.8), sondern genau um den Mittelpunkt des Kronkorkens (rotes Kreuz in Abbildung 10.8).

Für den `drawImage`-Befehl suchen wir also die Koordinaten der linken oberen Ecke des Kronkorkenausschnitts (blaues Kreuz in Abbildung 10.8). Ihre y-Koordinate[4] erhalten wir, indem wir (gemäß der Pfeile rechts in Abbildung 10.8)

1. vom Koordinatenursprung des Scans `y_roh` Bildpunkte nach unten zum Klickpunkt im Scan gehen
2. vom Klickpunkt im Scan 200 Bildpunkt nach oben an den oberen Rand des 400×400-Fensters gehen
3. vom oberen Rand des 400×400-Fensters `y_m/2` Bildpunkte nach unten zum Mittelpunkt des Kronkorkens gehen[5] und
4. vom Mittelpunkt des Kronkorkens 150 Bildpunkte nach oben an den oberen Rand des Kronkorkenausschnitts gehen.

[4] Die x-Koordinate der Ecke erhalten wir, indem wir jedes y durch x ersetzen.

[5] Wir müssen die Mittelpunktskoordinaten halbieren, da die Auflösung des Bildes, in dem wir den Mittelpunkt bestimmt haben, doppelt so groß ist wie die des Scans (Abschnitt 10.2).

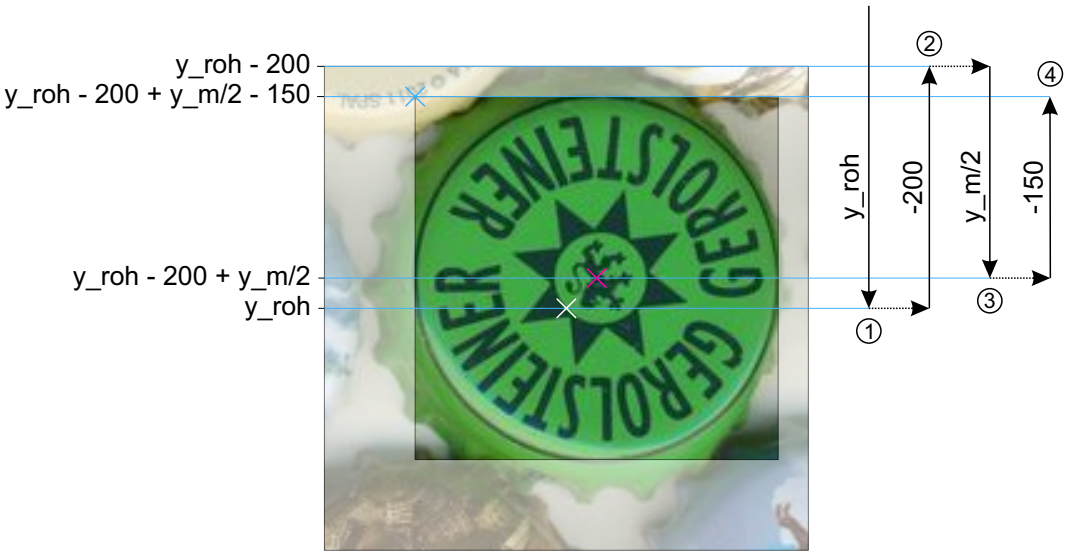

Abbildung 10.8: Weißes Kreuz: Klick im Scan
Rotes Kreuz: Mittelpunkt des Kronkorkens
Blaues Kreuz: Linke obere Ecke des Kronkorkens

Abschließend stellen wir noch sicher, dass die Leinwand mit dem 300×300 Bild unsichtbar ist:

```
leinwand_300.hidden = true
}
```

10.5 gitter_erzeugen

Wenn der Nutzer dies für die Feinkorrektur der Position und Lage des Bildes wünscht (Abschnitt 10.4.3), legen wir das in Abbildung 4.6 dargestellte Gitter über das Kronkorkenbild

```
function gitter_erzeugen() {
```

Dazu beginnen wir einen Linienzug

```
kontext_800.beginPath()
```

und eine Schleife, um die 19 waagerechten und 19 senkrechten Linien zu zeichnen:

```
for (var i = 1; i < 20; i += 1) {
```

Die waagerechten Linien beginnen jeweils am linken Rand des Bildes (`x_m - 300`)

```
kontext_800.moveTo(x_m - 300, y_m - 300 + 30 * i)
```

haben einen Abstand voneinander von 30 Bildpunkten (`y_m - 300 + 30 * i`) und enden am rechten Rand des Bildes (`x_m + 300`):

```
kontext_800.lineTo(x_m + 300, y_m - 300 + 30 * i)
```

Auf die gleiche Weise zeichnen wir die vertikalen Linien:

```
kontext_800.moveTo(x_m - 300 + 30 * i, y_m - 300)
kontext_800.lineTo(x_m - 300 + 30 * i, y_m + 300)
}
```

Dann begeben wir uns zum Mittelpunkt des Kronkorkens

```
kontext_800.moveTo(x_m, y_m)
```

und zeichnen dort 9 konzentrische Kreise (`arc`), wiederum mit einer jeweiligen Radiusvergrößerung von 30 Bildpunkten (`30 * i`):

```
for (var i = 1; i < 10; i += 1) {
    kontext_800.arc(x_m, y_m, 30 * i, 0, 2 * Math.PI, true)
}
```

Nicht vergessen dürfen wir abschließend den `stroke`-Befehl, ohne den das Gitter nicht gezeichnet werden würde:

```
kontext_800.stroke()
}
```

10.6 button_senden.onclick

Wenn der Nutzer die Schaltfläche Vergleich mit vorhandenen Kronkorken gedrückt hat (Abschnitt 4.3.3), senden wir das 300×300 Bild an den Server:

```
button_senden.onclick = function () {
```

Dazu verwenden wir den `toDataURL`-Befehl, der die Bilddaten in eine base64-kodierte[6] Zeichenkette wandelt

```
var bild_300 = leinwand_300.toDataURL("image/jpeg")
```

die dann beispielsweise so aussehen könnte:

```

```

Da wir später auf dem Server nur die Bildrohdaten verwenden wollen und natürlich wissen, dass sie ein base64-kodiertes jpg darstellen, entfernen wir vor der Übertragung den beschreibenden Kopf der Zeichenkette

[6] Das base64-Format benutzen wir, wenn wir binäre Daten (8 Bit) über ein Medium transportieren möchten, dass nicht alle 8-Bit-Zeichen übertragen kann. Dazu fassen wir jeweils drei 8-Bit-Zeichen zusammen und teilen diese 24 Bit dann wieder in vier 6-Bit-Zeichen ($2^6 = 64$ verschiedene Buchstaben, Zahlen und ein paar unkritische Zeichen) auf.

```
bild_300 = bild_300.replace("", "")
```

so dass nur die Rohdaten übrigbleiben:

```
S2xlaW5lciBTY2hsYXVtZWllciA7LSk=
```

Jetzt können wir einen jQuery-ajax-Befehl verwenden, um die Bilddaten zu senden

```
$.ajax({
```

dem wir die notwendigen Parameter als Name-Wert-Paare übergeben:

Als Anfragetyp verwenden wir `POST`, damit wir bei der Anfrage überhaupt Daten an den Server senden können:

```
type: "POST",
```

Die Anfrage schicken wir an einen Webdienst (Kapitel 11) auf dem Server

```
url: "bild_abspeichern.aspx/bild_abspeichern",
```

wobei wir die Bilddaten in der Variablen `bild_300`

```
data: '{"bild_300": "' + bild_300 + '" }',
```

im JSON-Format an den Server übertragen

```
contentType: "application/json; charset=utf-8",
```

Auch die Antwort des Servers (im Fehlerfall) erwarten wir als JSON-Objekt:

```
dataType: "json",
```

Wenn die Bilddaten vom Server angenommen wurden, wechseln wir sofort auf die Dublettenerkennungsseite (Kapitel 12):

```
success: function (response, nachricht) {
    location.href = "vergleichen.aspx"
},
```

Im Fehlerfall informieren wir den Nutzer darüber, was schiefgegangen ist:

```
error: function (response, nachricht) {
  alert("error: " + response.responseText + nachricht + response
     .s + response.d)
} }); }
```

10.7 taste_gedrueckt

Wenn der Nutzer eine Taste gedrückt hat (Abschnitt 4.3.3)

```
<body onkeydown="taste_gedrueckt(event)">
```

rufen wir das entsprechende Unterprogramm auf:

```
function taste_gedrueckt(event) {
```

In Abhängigkeit von der gedrückten Taste

```
if (event.keyCode == 65) { // a
```

verschieben wir das Bild, indem wir die Koordinaten seines Klickpunktes um jeweils eine Bildpunkt verschieben:

```
x_roh += Math.cos(phi)
y_roh -= Math.sin(phi)
```

Dabei müssen wir allerdings die Verschiebung in das (möglicherweise) gedrehte Koordinatensystem transformieren (`cos(phi)`, `sin(phi)`), damit die Positionsänderung auf dem Bildschirm horizontal bzw. vertikal verläuft. Anschließend zeichnen wir das Bild neu:

```
zeichnen()
}
```

Alle Verschiebungstastenaktionen (`w a s d`) sind ähnlich aufgebaut:

```
else if (event.keyCode == 68) { // d
    x_roh -= Math.cos(phi)
    y_roh += Math.sin(phi)
    zeichnen()
}

else if (event.keyCode == 83) { // s
    x_roh -= Math.sin(phi)
    y_roh -= Math.cos(phi)
    zeichnen()
}

else if (event.keyCode == 87) { // w
    x_roh += Math.sin(phi)
    y_roh += Math.cos(phi)
    zeichnen()
}
```

Die Tasten `q` und `e` rotieren das Bild um einen Winkel von $0.005\,\text{rad} \approx 0.3°$ nach links bzw. rechts:

```
else if (event.keyCode == 81) { // q
    phi -= 0.005
    zeichnen()
```

```
}

else if (event.keyCode == 69) { // e
    phi += 0.005
    zeichnen()
}
```

Mit der Taste g schalten wir das Gitter ein bzw. aus (Abschnitt 10.4.3):

```
else if (event.keyCode == 71) { // g
    gitter_an = !gitter_an
    zeichnen()
}}
```

11 bild_abspeichern.aspx

Wenn der Nutzer die Schaltfläche Vergleich mit vorhandenen Kronkorken gedrückt hat (Abschnitt 4.3.3), senden wir das 300 × 300 Bild an den Server (Abschnitt 10.6). Auf der Serverseite nimmt ein Webdienst (`WebMethod`) das Bild entgegen und speichert es als 300 × 300- und als 40 × 40 Bild ab:

```
<System.Web.Services.WebMethod()> _
Public Shared Sub bild_abspeichern(bild_300 As String)
```

Dazu legen wir mit Hilfe eines `FileStreams` die Datei `kandidat_300.jpg` an, in der wir das 300 × 300 Bild abspeichern werden:

```
Dim fs As New FileStream(HttpContext.Current.Request.MapPath
   ("kandidat_300.jpg"), FileMode.Create)
```

Mit einem `BinaryWriter` können wir Bytefelder binär in einen Stream schreiben:

```
Dim bw As New BinaryWriter(fs)
```

Dazu wandeln wir das im Base64-Format kodierte Bild zurück in ein Bytefeld

```
Dim data As Byte() = Convert.FromBase64String(bild_300)
```

und schreiben es mit Hilfe des BinaryWriters über den Stream in die Datei:

```
bw.Write(data)
```

Jetzt müssen wir nur noch wir den Zugriff auf die Datei schließen, um beim nächsten Aufruf des Webdienstes wieder in die gleiche Datei schreiben zu können:

```
bw.Close()
fs.Close()
```

In den restlichen Zeilen des Webdienstes skalieren wir das 300 × 300 Bild auf 40 × 40 Bildpunkte herunter und speichern es ab. Dazu lesen wir das 300 × 300 Bild in eine Bitmap

```
Dim bm As New Bitmap(HttpContext.Current.Request.MapPath("
   kandidat_300.jpg"))
```

und definieren eine neue 40 × 40-Bitmap:

```
Dim bm_neu As New Bitmap(40, 40)
```

Um die 40×40-Bitmap beschreiben zu können, definieren wir darauf eine GDI+-Zeichnungsoberfläche

```
Dim g As Graphics = Graphics.FromImage(bm_neu)
```

mit folgenden Eigenschaften, die für ein optimales Herunterskalieren sorgen:

```
g.InterpolationMode = InterpolationMode.HighQualityBicubic
g.SmoothingMode = SmoothingMode.HighQuality
g.PixelOffsetMode = PixelOffsetMode.HighQuality
g.CompositingQuality = CompositingQuality.HighQuality
g.CompositingMode = CompositingMode.SourceOver
```

Zum Herunterskalieren selbst zeichnen wir das 300×300 Bild in ein 40×40 Bildpunkte großes Rechteck auf der Zeichenoberfläche:

```
g.DrawImage(bm, New Rectangle(0, 0, 40, 40), New Rectangle
   (0, 0, 300, 300), GraphicsUnit.Pixel)
```

Jetzt können wir das 40×40 Bild in die Datei `kandidat_40.jpg` abspeichern

```
bm_neu.Save(HttpContext.Current.Request.MapPath("kandidat_40
   .jpg"), ImageFormat.Jpeg)
```

und den Speicherplatz der Bilder wieder freigeben:

```
bm.Dispose()
bm_neu.Dispose()
```

12 vergleichen.aspx

Auf dieser Seite (Abbildung 4.8) vergleichen wir den aktuellen Kandidaten mit allen in der Datenbank vorhandenen Kronkorken und stellen die (momentan 100) ähnlichsten Dublettenkandidaten dar.

12.1 Page_load

Als erstes überprüfen wir, ob der Nutzer angemeldet ist und schicken ihn ansonsten zur Anmeldung auf die „erste" Seite (Kapitel 9) der Kronkorkenerfassung:

```
If Not Session("angemeldet") Then
    Response.Redirect("kronkorken_auswaehlen.aspx")
End If
```

Als nächstes untersuchen wir, ob schon alle bislang in den Bestand aufgenommenen Muster zur Dublettenerkennung eingelesen wurden. Wenn nicht, holen wir dies durch Aufruf von `bilder_lesen.aspx` (Kapitel 13) nach:[1]

```
If IsNothing(Session("bytes")) Then
    Response.Redirect("bilder_lesen.aspx")
End If
```

Das Unterprogramm `bmp2byte` (Abschnitt 12.3) liefert uns die Farbtöne aller Bildpunktes des aktuellen Kandidaten als Bytefeld zurück:

```
Dim kandidat = bmp2byte()
```

Die Farbtöne aller Bildpunkte der schon erfassten Kronkorken, mit denen wir die des Kandidaten vergleichen wollen, speichern wir in der Variablen `bytes` zwischen:

```
Dim bytes As Byte() = Session("bytes")
```

[1] Das Einlesen tausender Bilder dauert auf aktuellen Servern schon ein paar Sekunden. Wir wiederholen dies daher nicht bei jedem Erkennungsprozess, sondern speichern die eingelesenen Muster zur direkten Weiterverwendung in einer Sessionvariablen ab. Dies hat zur Folge, dass die jeweils aktuell kurz vorher aufgenommenen Kronkorken nicht zum Vergleich herangezogen werden, was eigentlich kein Problem sein sollte, da wir uns wenigstens noch daran erinnern können sollten, welche Kronkorken wir in der aktuellen Session gerade eben erfasst haben. Alternativ können wir uns kurz ab- und wieder anmelden oder die Seite `bilder_lesen.aspx` direkt manuell aufrufen, um mit wirklich allen momentan erfassten Kronkorken vergleichen zu können.

Auf der Seite bilder_lesen.aspx (Kapitel 13) haben wir außerdem die ids aller vorhandenen Kronkorken in einer Sessionvariable gespeichert. Da wir das Id-Feld später sortieren werden, speichern wir die Ids erst einmal zwischen

```
Dim ids_buffer As String() = Session("ids")
```

bevor wir das eigentliche Arbeitsfeld deklarieren

```
Dim ids(ids_buffer.Count - 1) As String
```

und aus dem Zwischenspeicher füllen:

```
ids_buffer.CopyTo(ids, 0)
```

In der Variable n_ds speichern wir die Anzahl der bislang erfassten Kronkorken:

```
Dim n_ids = ids.Count
```

Für einen selbstsprechenden Programmierstil definieren wir die Anzahl der Zeilen, Spalten und Farbbytes (RGB) des Kronkorkens als Konstanten

```
Dim n_zeilen = 40
Dim n_spalten = 40
Dim n_farben = 3
```

und berechnen daraus die Anzahl der Bytes pro Zeile und pro Bild:

```
Dim n_bytes_pro_zeile = n_spalten * n_farben
Dim n_bytes_pro_bild = n_zeilen * n_bytes_pro_zeile
```

Wir definieren den Radius, innerhalb dessen die Bildpunkte liegen, die wir zur Dublettenerkennung heranziehen wollen (Abschnitt 4.4.1)

```
Dim radius = 18
```

und berechnen auch gleich sein Quadrat, das wir im Folgenden mehrfach verwenden werden:

```
Dim radius_quadrat = radius * radius
```

12.1.1 Mittlerer Farbton

Als erstes wollen wir jetzt den mittleren Farbton (Abschnitt 4.4.2) des Kandidaten ermitteln, um ihn gegebenenfalls als eine Eigenschaft des neuen Kronkorkens in der Datenbank abspeichern zu können.[2]

Dazu initialisieren wir die aufsummierten Rot-, Grün- und Blauwerte und die Anzahl der verarbeiteten Bildpunkte

[2]Der Farbton (Hue) stellt ein Kriterium dar, nach dem wir Kronkorken sortieren lassen können (Titelbild, Abschnitt 2.3 und Abschnitt 7.3).

```
Dim rot_summe = 0
Dim gruen_summe = 0
Dim blau_summe = 0
Dim n_pixel = 0
```

und starten eine Doppelschleife über alle Bildpunkte, die mit einer äußeren Schleife über alle Zeilen beginnt:

```
For i_zeile = 0 To n_zeilen - 1
```

Innerhalb der Zeilenschleife berechnen wir den quadratischen Abstand der aktuellen Zeile von der Zeile des Bildmittelpunktes, den wir später benutzen werden:

```
Dim zeilen_quadrat = (i_zeile - 20) * (i_zeile - 20)
```

Auch in der inneren Schleife über alle Spalten

```
For i_spalte = 0 To n_spalten - 1
```

berechnen wir als erstes das Spaltenabstandsquadrat:

```
Dim spalten_quadrat = (i_spalte - 20) * (i_spalte - 20)
```

Da wir nur Bildpunkte innerhalb eines Mittelpunktkreises mit dem Radius $R = 18$ zur Charakterisierung eines Kronkorkens heranziehen wollen (Abschnitt 4.4.1), untersuchen wir jetzt – wie in Abbildung 12.1 dargestellt – ob der pythagoreische Abstand r des Punktes vom Mittelpunkt kleiner ist als der vorgegebene Radius R:

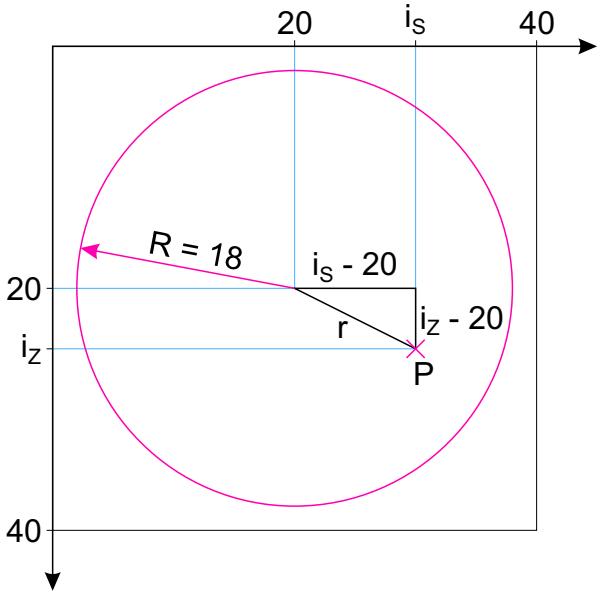

Abbildung 12.1: Liegt der Punkt P innerhalb des Kreises mit dem Radius $R = 18$? Ist also $r^2 = (i_S - 20)^2 + (i_z - 20)^2 \leq R^2$?

Wenn dies der Fall ist

```
If zeilen_quadrat + spalten_quadrat <= radius_quadrat Then
```

dann bestimmen wir den Index des Bildpunktes[3]

```
Dim i_pixel = i_spalte + i_zeile * n_spalten
```

und den Index der Farbwerte, der berücksichtigt, dass jeder Bildpunkt drei Farbwerte besitzt:

```
Dim i_kandidat = i_pixel * n_farben
```

Beim Lesen der Farbwerte in Abschnitt 12.3 werden diese nicht im üblichen RGB-Format (Reihenfolge Rot → Grün → Blau), sondern in der umgekehrten Reihenfolge BGR (Blau → Grün → Rot) abgelegt, so dass der rote Farbwert zwei Indexstellen (`i_kandidat + 2`) hinter dem blauen liegt:[4]

```
rot_summe += kandidat(i_kandidat + 2)
gruen_summe += kandidat(i_kandidat + 1)
blau_summe += kandidat(i_kandidat + 0)
```

Durch das Aufaddieren (`+=`) enthalten die Variablen `rot_summe`, `gruen_summe` und `blau_summe` nach Beendigung der Schleife die Farbwertesummen aller Bildpunkte. Wenn wir jetzt auch noch die Anzahl der Bildpunkte hoch zählen

```
n_pixel += 1
```

können wir nach Beendigung der Schleife

```
End If
Next
Next
```

die mittlere Farbe eines Kronkorkens bestimmen, indem wir jeweils die Farbwertesummen durch die Gesamtanzahl der Bildpunkte teilen:

```
Dim farbe_kandidat_mittel As Color = _
  Color.FromArgb( _
    255, _
    rot_summe / n_pixel, _
    gruen_summe / n_pixel, _
    blau_summe / n_pixel)
```

[3] Die Bildpunkte werden zeilenweise im Bytefeld abgelegt.

[4] Erschwerend kommt hinzu, dass aus Speicherverwaltungseffizienzgründen die interne Anzahl der Bytes pro Bildzeile ein Vielfaches von vier sein muss. Ist dies nicht der Fall, wird eine Bildzeile am Ende automatisch mit entsprechend vielen Nullen aufgefüllt. Da unsere Bilder allerdings immer eine Zeilenlänge von 40 bzw. 300 Bildpunkten haben, die ganzzahlig durch vier teilbar ist, müssen wir dieses Problem glücklicherweise nicht berücksichtigen.

Dabei erwartet die Methode `FromArgb` als ersten Parameter noch den Wert des Alphakanals, der die Deckkraft der Farbe angibt und der bei allen unseren Kronkorkenbildern den maximalen Wert von 255 besitzt.

Wie in Abschnitt 4.4.2 beschrieben, interessiert uns dann für die Farbreihenfolge der mittlere Hue-Wert des Kronkorkens:

```
Session("hue") = CInt(farbe_kandidat_mittel.GetHue)
```

12.1.2 Vergleich aller Bildpunkte

In diesem Abschnitt wollen wir jeden einzelnen Bildpunkt des Kandidaten mit den entsprechenden Bildpunkten aller schon in der Datenbank vorhandenen Kronkorken vergleichen, um über den mittleren Farbraumabstand des Kandidaten von den vorhandenen Kronkorken eine Auswahl der dem Kandidaten am ähnlichsten Kronkorken zu erhalten.

Dazu deklarieren wir ein Feld, in das wir später die mittleren Farbraumabstände jedes Kronkorkens vom Kandidaten eintragen werden:

```
Dim distances(n_ids - 1) As ULong
```

Jetzt folgen drei ineinander geschachtelte Schleifen. Die äußere Schleife läuft über alle schon vorhandenen Kronkorken:

```
For i_ids = 0 To n_ids - 1
```

Bei jedem Schleifendurchlauf setzen wir die Variable, die den Abstand des aktuellen Kronkorkens vom Kandidaten beinhaltet, zurück:

```
Dim distance As ULong = 0
```

Die inneren beiden Schleifen laufen analog zu Abschnitt 12.1.1 durch alle Bildpunkte innerhalb des Kreises mit dem Radius $R = 18$.

```
For i_zeile = 0 To n_zeilen - 1

Dim zeilen_quadrat = (i_zeile - 20) * (i_zeile - 20)

For i_spalte = 0 To n_spalten - 1

Dim spalten_quadrat = (i_spalte - 20) * (i_spalte - 20)

If zeilen_quadrat + spalten_quadrat <= radius_quadrat Then

Dim i_pixel = i_spalte + i_zeile * n_spalten
```

Wie in Abschnitt 4.4.2 beschrieben, hat der Nutzer die Möglichkeit, entweder nur den Farbton (Hue) oder alle Grundfarben (RGB) zum Vergleich heranzuziehen. Wenn er also in Abbildung 4.8 Farbige Kronkorken ausgewählt hat,

```
If RadioButtonList_suchalgorithmus.SelectedValue = "hue"
   Then
```

bestimmen wir analog[5]zu Abschnitt 12.1.1 den Farbton des aktuellen Bildpunktes des Kandidaten:

```
Dim i_kandidat = i_pixel * n_farben

Dim farbe_kandidat As Color = _
    Color.FromArgb( _
        255, _
        kandidat(i_kandidat + 2), _
        kandidat(i_kandidat + 1), _
        kandidat(i_kandidat + 0))

Dim hue_kandidat = farbe_kandidat.GetHue
```

und des aktuellen Kronkorkens:

```
Dim i_byte = i_kandidat + i_ids * n_bytes_pro_bild

Dim farbe_byte As Color = _
    Color.FromArgb( _
        255, _
        bytes(i_byte + 2), _
        bytes(i_byte + 1), _
        bytes(i_byte + 0))

Dim hue_byte = farbe_byte.GetHue
```

Den farblichen Unterschied zwischen beiden Bildpunkten berechnen wir dann über den Betrag der Differenz beider Farbtöne:

```
Dim hue_distance = Math.Abs(hue_kandidat - hue_byte)
```

Dabei gibt es ein kleines Problem: Die Farbtöne sind wie die Gradzahlen eines Kompasses auf dem Umfang eines Kreises angeordnet. Wie in Abbildung 12.2 ersichtlich, entspricht ein Farbton von 0 der Farbe rot, ein Farbton von 120 steht für die Farbe grün, 240 für blau und 360 theoretisch wieder für rot. Wenn wir nun den Abstand der Farbe rot (mit einem Farbton von 0) von der Farbe blau (mit einem Farbton von 240) bestimmen, erhalten wir als Farbtondifferenz den Wert 240:

$$h_{blau} - h_{rot} = 240 - 0 = 240$$

[5]Im Unterschied zu Abschnitt 12.1.1 bestimmen wir hier nicht den mittleren Farbton des gesamten Kronkorkens (`rot_summe / n_pixel` ...), sondern den Farbton eines einzelnen Bildpunktes.

Andererseits ist nicht einzusehen, warum blau von rot doppelt so weit entfernt sein sollte wie grün:

$$h_{grün} - h_{rot} = 120 - 0 = 120$$

Statt also von rot den langen Weg links herum gegen den Uhrzeigersinn bis blau zu gehen, könnten wir doch auch rechts herum, im Uhrzeigersinn den kurzen Weg bis blau gehen; dann wäre die Entfernung zwischen blau und rot auch nur 120.[6]

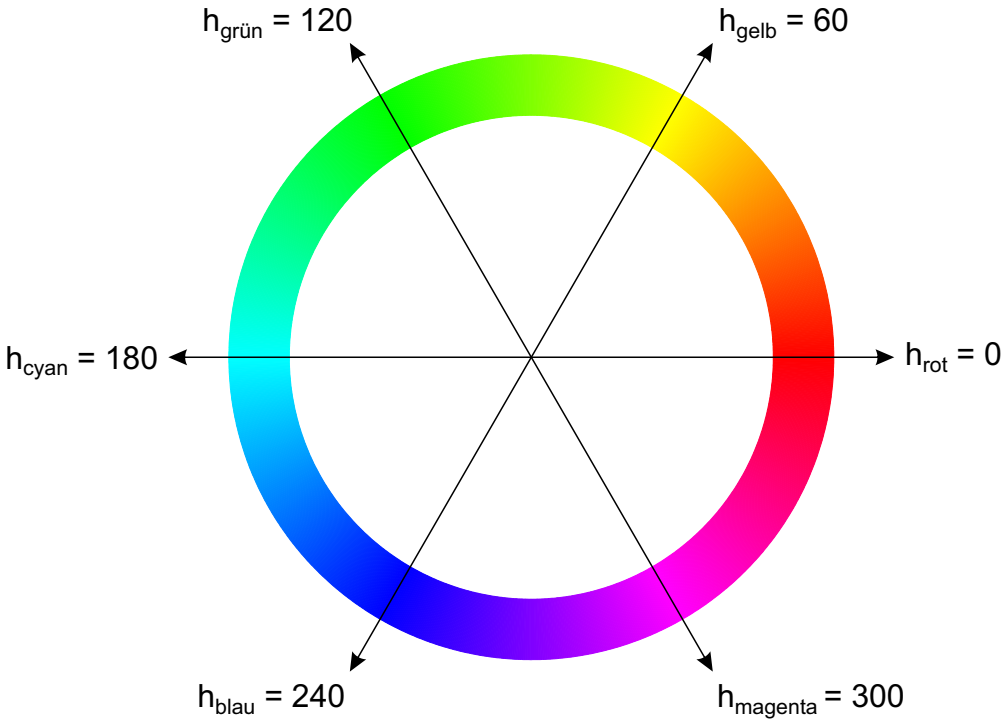

Abbildung 12.2: Farbtonring

Um nun immer den kürzesten Weg zu gehen, ziehen wir die Farbtondifferenz einfach von 360 ab, wenn sie größer als 180 geworden ist:

```
If hue_distance > 180 Then

    hue_distance = 360 - hue_distance

End If
```

Auf diese Weise beträgt beispielsweise der Abstand zwischen blau und rot jetzt nicht mehr 240, sondern nur noch wie gewünscht $360 - 240 = 120$.

[6] Strenggenommen ist es sogar schwierig zu begründen, dass cyan weiter von rot entfernt ist als blau oder grün, denn schließlich enthalten alle drei Farben keinen Rotanteil. Aber – in irgendeiner geschlossenen Reihenfolge müssen wir die Farben schließlich anordnen ...

Wenn wir nun die Unterschiede aller Bildpunktefarbtöne aufaddieren, erhalten wir am Ende der inneren Doppelschleife ein Maß für den Unterschied zwischen dem Kandidaten und dem aktuellen Kronkorken:

```
distance += hue_distance
```

Falls der Nutzer in Abschnitt 4.4.2 die Schaltfläche Graue Kronkorken gedrückt hat, verwenden wir statt des Farbtons die drei RGB-Komponenten aller Bildpunkte zur Unterscheidung:

```
Else
```

Dazu starten wir eine innerste Schleife über die drei Farbkomponenten:

```
For i_farbe = 0 To n_farben - 1
```

Da die RGB-Komponenten eines Bildpunktes im Bytefeld direkt hintereinander liegen, können wir sehr einfach alle Farbkomponenten aller Bildpunkte des Kandidaten nacheinander ansprechen:

```
Dim i_kandidat = i_farbe + i_pixel * n_farben
```

Die entsprechenden Farbkomponenten des aktuellen Kronkorkens erhalten wir, wenn wir zusätzlich noch mit `i_ids * n_bytes_pro_bild` jeweils einen ganzen Kronkorken „weiterspringen”:

```
Dim i_byte = i_kandidat + i_ids * n_bytes_pro_bild
```

Der Gesamtunterschied zwischen aktuellem Kronkorken und Kandidat ergibt sich dann als Summe der absoluten Differenzen entsprechender Byte-Werte:

```
distance += Math.Abs(CInt(bytes(i_byte)) - CInt(kandidat(
   i_kandidat)))

Next
End If
End If
Next
Next
```

Unabhängig davon, welche Schaltfläche der Nutzer gedrückt hat, speichern wir den aktuellen Unterschied in der entsprechenden Zelle des Abstandsfeldes ab:

```
distances(i_ids) = distance

Next
```

Da wir dem Nutzer die 100 Kronkorken präsentieren wollen, die dem Kandidaten am ähnlichsten sehen, lassen wir das Abstandsfeld sortieren, so dass die Kronkorken mit den

kleinsten Unterschieden oben stehen. Freundlicherweise können wir dabei auch gleich das Feld, in dem wir die Ids der Kronkorken abgespeichert hatten, entsprechend der Abstände mitsortieren lassen:

```
Array.Sort(distances, ids)
```

Jetzt können wir eine Schleife über die ersten 100 Kronkorken laufen lassen

```
For i_id = 0 To 99
```

in der wir jeweils die Id aus dem mitsortierten Feld lesen

```
Dim aktuelle_id = ids(i_id)
```

und für jeden Kronkorken einen neuen ImageButton erzeugen

```
Dim image_button As New System.Web.UI.WebControls.
   ImageButton
```

dessen Bild wir über seine Id als Dateinamen aus dem 300 × 300-Verzeichnis lesen:

```
image_button.ImageUrl = "300x300\" & aktuelle_id & ".jpg"
```

Um die 100 Kronkorken auch auf Monitoren mit geringerer Auflösung ohne viel Scrollen darstellen zu können, skalieren wir sie auf 100 × 100 Bildpunkte herunter:[7]

```
image_button.Height = 100
image_button.Width = 100
```

Anschließend definieren wir das Unterprogramm `image_button_Click` (Abschnitt 12.2), das aufgerufen wird, wenn der Nutzer auf einer abgebildeten Kronkorken klickt

```
AddHandler image_button.Click, AddressOf image_button_Click
```

und verstecken die Id des Kronkorkens im Alternativtext des Bildes

```
image_button.AlternateText = aktuelle_id
```

damit wir sie in Abschnitt 12.2 verwenden können. Abschließend fügen wir das Kronkorkenbild noch dem vordefinierten Panel hinzu, damit es angezeigt wird:

```
Panel_image_buttons.Controls.Add(image_button)
```

[7] 100 × 100 Bildpunkte sind ein Kompromiss. 100 Kronkorken der Auflösung 300 × 300 würden zu viel Platz auf dem Bildschirm benötigen; Bilder mit 40 × 40 Bildpunkten würden nicht ausreichen, um Unterschiede kleinerer Details zu erkennen.

12.2 image_button_click

Dieses Unterprogramm wird immer dann aufgerufen, wenn der Nutzer auf das Bild einer potenziellen Dublette geklickt hat. Wir lesen dann die Id des Kronkorkens aus dem Alternativtext des Bildes und rufen direkt die Detailseite des Kronkorkens auf:

```
Response.Redirect("cap.aspx?id=" & sender.AlternateText)
```

12.3 bmp2byte

Die Aufgabe von bmp2byteist es, eine Bilddatei von der Platte zu lesen und seine Farbwerte in Form eines Bytefeldes zurück zu geben, um die Farbwerte des Kandidaten in Abschnitt 12.1 mit denen der übrigen Kronkorken zu vergleichen. Dazu lesen wir die Bilddatei in ein neues Bitmap-Objekt

```
Dim bmp As New Bitmap(MapPath("kandidat_40.jpg"))
```

definieren ein Rechteck mit den Abmessungen des Bildes

```
Dim rect As New Rectangle(0, 0, bmp.Width, bmp.Height)
```

und sperren die binären Daten der Bitmap mit Hilfe des LockBits-Befehls im Systemspeicher:

```
Dim bmpData As BitmapData = bmp.LockBits( _
  rect, _
    ImageLockMode.ReadOnly, _
    bmp.PixelFormat)
```

Der Befehl Scan0 liefert uns einen Zeiger auf die interne Adresse der ersten Zeile des Bildes

```
Dim ptr As IntPtr = bmpData.Scan0
```

während der Befehl Stride die Länge einer Bildzeile zurück gibt:[8]

```
Dim bytes As Integer = Math.Abs(bmpData.Stride) * bmp.Height
```

Mit diesen Informationen dimensionieren wir ein Bytefeld, das die Farbwerte aller Bildpunkte des Kandidaten beinhalten wird:

```
Dim rgbValues(bytes - 1) As Byte
```

Der Marshal.Copy-Befehl kann nun den gesamten Systemspeicherbereich, in dem sich das Bild befindet, in einem „Rutsch” in das Bytefeld kopieren, was um Größenordnungen schneller geschieht als das byteweise Kopieren der einzelnen Bildpunkte:

[8] Wir verwenden in dieser allgemeinen Form den Betrag der Zeilenlänge, da diese negativ wäre, wenn das Bild von unten nach oben aufgebaut sein würde.

```
System.Runtime.InteropServices.Marshal.Copy(ptr, rgbValues,
   0, bytes)
```

Sinnvollerweise entsperren wir jetzt noch wieder die Bilddaten im Systemspeicher

```
bmp.UnlockBits(bmpData)
```

und geben den nicht mehr benötigten Speicher des Bildes wieder frei:

```
bmp.Dispose()
```

Als Rückgabewert des Unterprogramms verwenden wir nun das mit dem Bilddaten gefüllte Bytefeld:

```
Return rgbValues
```

12.4 Button_speichern_Click

Wenn der Nutzer auf die Schaltfläche In die Datenbank aufnehmen klickt, um den Kandidaten in die Datenbank aufzunehmen, laden wir die Datenbank

```
Dim kronkorken_datei = MapPath("kronkorken.xml")
Dim kronkorken_xml = XElement.Load(kronkorken_datei)
```

und definieren im folgenden einen neuen Eintrag für den Kandidaten. Dazu erzeugen wir eine neue Id

```
Dim neue_id = System.Guid.NewGuid.ToString()
```

und verwenden dann LINQ (Language Integrated Query), um den Eintrag strukturell genau so aufzubauen, wie er später in der XML-Datei stehen wird:

```
<kronkorken>
  <id>
    <%= neue_id %>
  </id>
  <getraenk>
    <%= Session("getraenk") %>
  </getraenk>
  :
  <tauschen>
    <%= 0 %>
  </tauschen>
  <von>
    <%= Session("von") %>
  </von>
</kronkorken>
```

Dabei entnehmen wir die Voreinstellung der neuen Kronkorkeneigenschaften (getraenk, marke, ... von) jeweils der entsprechenden Sessionvariablen. Dies bedeutet in vielen Fällen eine enorme Arbeitserleichterung, da neue Kronkorken – wie in Abschnitt 4.4.3 beschrieben – häufig in vielen Eigenschaften mit schon vorhandenen übereinstimmen und die Sessionvariablen bei jedem Aufruf der Detailseite eines Kronkorkens gesetzt werden (Abschnitt 8.1).

Den neuen Datenbankeintrag hängen wir dann hinten an die vorher geladene XML-Struktur

```
kronkorken_xml.Add(neu_kronkorken_xml)
```

und speichern diese wieder in die XML-Datei ab:

```
kronkorken_xml.Save(kronkorken_datei)
```

Die Bilder des Kandidaten kopieren wir in beiden Auflösungen in die entsprechenden Verzeichnisse:

```
FileCopy(MapPath("kandidat _300.jpg"), MapPath("300x300/" &
   neue_id & ".jpg"))

FileCopy(MapPath("kandidat_40.jpg"), MapPath("40x40/" &
   neue_id & ".jpg"))
```

Abschließend wechseln wir auf die Detailseite des neu aufgenommenen Kronkorkens, um seine individuellen Eigenschaften einzutragen:

```
Response.Redirect("cap.aspx?id=" & neue_id)
```

12.5 Button_zurueck_Click

Ein Klick auf die Schaltfläche Zurück zur Auswahl führt uns natürlich wieder zurück zur Auswahlseite:

```
Response.Redirect("kronkorken_auswaehlen.aspx")
```

13 bilder_lesen.aspx

Die Seite `bilder_lesen.aspx` wird von der Seite `vergleichen.aspx` einmalig aufgerufen (Abschnitt 12.1), um alle schon vorhandenen 40×40 Kronkorkenbilder einzulesen und – zusammen mit ihren Ids – in einem Bytefeld abzuspeichern.

13.1 Page_load

Wir lesen die Kronkorkendatenbank

```
Dim kronkorken_datei = MapPath("kronkorken.xml")
Dim kronkorken_xml = XElement.Load(kronkorken_datei)
```

um an die Liste aller Ids heranzukommen:[1]

```
Dim alle_ids =
  From kronkorken In kronkorken_xml.<kronkorken>
  Select
  id = kronkorken.<id>.Value
  Where id <> "42"
```

Wir ermitteln die Anzahl der vorhandenen Kronkorken

```
Dim n_ids = alle_ids.Count
```

definieren die Anzahl der Bytes pro Bild

```
Dim n_bytes_pro_bild = 40 * 40 * 3
```

und dimensionieren ein Zeichenkettenfeld, das die Ids der Kronkorken aufnehmen wird

```
Dim ids(n_ids - 1) As String
```

und ein Bytefeld für die Binärdaten der Bilder

```
Dim bytes(n_ids * n_bytes_pro_bild - 1) As Byte
```

In der anschließenden Schleife über alle Kronkorken

[1] Dabei müssen wir den ersten Kronkorken mit der Id von 42 ignorieren, da dieser in der Datenbank nur existiert (Kapitel 6), damit wir in den automatisch aus der XML-Datei erzeugten Auswahllisten auf der Übersichtsseite (Abbildung 2.1) auch leere Einträge anwählen können.

```
For i_ids = 0 To n_ids - 1
```

lesen wir die Binärdaten eines einzelnen Kronkorkens (Abschnitt 13.2) und hängen diese hinten an das Bytefeld an:

```
bmp2byte(alle_ids(i_ids)).CopyTo(bytes, i_ids *
   n_bytes_pro_bild)
```

Außerdem fügen wir die Id des aktuellen Kronkorkens an das Ende des Zeichenkettenfeldes an:

```
ids(i_ids) = alle_ids(i_ids)

Next
```

Nach Beenden der Schleife speichern wir beide Felder in Sessionvariablen

```
Session("ids") = ids
Session("bytes") = bytes
```

und springen zurück zur Vergleichsseite:

```
Response.Redirect("vergleichen.aspx")
```

13.2 bmp2byte

Den Aufbau und die Funktionsweise dieses Unterprogrammes haben wir schon in Abschnitt 12.3 beschrieben. Als einzige Erweiterung übergeben wir hier noch die Id des Kronkorkens, dessen Binärdaten wir einlesen wollen:

```
Function bmp2byte(id As String) As Byte()

Dim bmp As New Bitmap(MapPath("40x40\" & id & ".jpg"))
```

A Mittelpunktkoordinaten eines durch drei Punkte definierten Kreises

Durch drei Punkte – die nicht auf einer Geraden liegen – ist ein Kreis definiert.

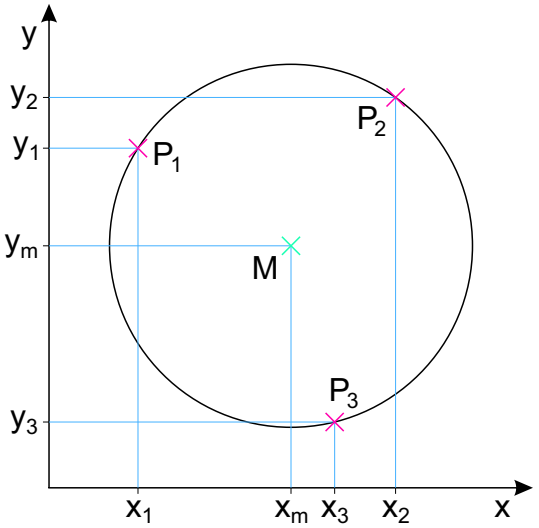

Abbildung A.1: Kreis aus drei Punkten berechnen

Wir können also aus den x- und y-Koordinaten der drei in Abbildung A.1 rot markierten Punkte den Radius und die Koordinaten des grün gekennzeichneten Mittelpunktes des Kreises berechnen. Da dazu die Lösung eines nichtlinearen Gleichungssystems bestehend aus drei quadratischen Gleichungen notwendig ist, verwenden wir ein symbolisches Matlab-Programm, in dem wir als erstes alle Variablen, die erstmalig auf der rechten Seite eines Gleichheitszeichens auftreten, als symbolisch deklarieren:[1]

```
syms x y          % Allgemeine Koordinaten
syms x1 x2 x3     % x-Koordinaten der drei Punkte
syms y1 y2 y3     % y-Koordinaten der drei Punkte
syms xm ym        % Mittelpunktskoordinaten des Kreises
syms r positive % Radius des Kreises
```

[1] Um die bei der Lösung der quadratischen Gleichungen auftretenden negativen Radien zu vermeiden, deklarieren wir den Radius dabei gleich als positiv.

Als nächstes definieren wir eine allgemeine Kreisgleichung in symbolischer Form:[2]

```
kreis = (x - xm)^2 + (y - ym)^2 == r^2

kreis =

(x - xm)^2 + (y - ym)^2 == r^2
```

Jetzt setzen wir mit dem `subs`-Befehl die drei Punkte einzeln in die Kreisgleichung ein und speichern die drei Bestimmungsgleichungen in den Variablen `k1`, `k2` und `k3`:

```
k1 = subs (kreis, {x, y}, {x1, y1})

k1 =

(x1 - xm)^2 + (y1 - ym)^2 == r^2

k2 = subs (kreis, {x, y}, {x2, y2})

k2 =

(x2 - xm)^2 + (y2 - ym)^2 == r^2

k3 = subs (kreis, {x, y}, {x3, y3})

k3 =

(x3 - xm)^2 + (y3 - ym)^2 == r^2
```

Der `solve`-Befehl bekommt dann als Parameter die drei Bestimmungsgleichungen und die drei Unbekannten, nach denen er auflösen soll:[3]

```
[~, xm, ym] = solve (k1, k2, k3, r, xm, ym);
```

Mit dem `pretty`-Befehl lassen sich die Mittelpunktskoordinaten schließlich halbwegs übersichtlich ausgeben:

```
pretty (xm)
```

```
     2          2          2          2          2          2
  (x1  y2 - x1  y3 - x2  y1 + x2  y3 + x3  y1 - x3  y2 +

       2          2            2          2      2            2
     y1  y2 - y1  y3 - y1 y2  + y1 y3  + y2  y3 - y2 y3 ) /
```

[2] Da wir den Befehl `echo on` verwendet haben, gibt Matlab sowohl den von uns eingegebenen Originalbefehl als auch das „Ergebnis der Berechnung" aus. In dieser reinen Definition stimmen beide natürlich überein.

[3] Das Semikolon am Ende der Zeile bewirkt, dass die Lösungen nicht ausgegeben werden. Die Tilde (`~`) bedeutet, dass der Radius zwar eine Unbekannte ist, aber nicht explizit ausgerechnet werden muss.

```
     (2 (x1 y2 - x2 y1 - x1 y3 + x3 y1 + x2 y3 - x3 y2))
```

```
pretty (ym)
```

```
       2         2            2         2         2         2
  (- x1  x2 + x1  x3 + x1 x2  - x1 x3  + x1 y2  - x1 y3  -

       2            2         2         2         2         2
     x2  x3 + x2 x3  - x2 y1  + x2 y3  + x3 y1  - x3 y2 ) /

     (2 (x1 y2 - x2 y1 - x1 y3 + x3 y1 + x2 y3 - x3 y2))
```

Noch etwas übersichtlicher können wir das Ergebnis mit dem `cute`-Befehl [4]

```
cute ([xm; ym])
```

ausgeben (Abbildung A.2)

Abbildung A.2: Symbolische Matlabausgabe [4] via MathJax [5] in Browserfenster

Literaturverzeichnis

[1] J. J. Buchholz. (2014) Suse's crown caps. [Online]. Available: http://suse.thebuchis.de/crown/

[2] J. Solero, D. Mascherini, and W. Veld. (2014) Kronkorkensammler. [Online]. Available: http://www.crowncaps.info/phplib/cci_members.php

[3] S. Lee-Delisle. (2014) Drawing rotated images into canvas. [Online]. Available: http://creativejs.com/2012/01/day-10-drawing-rotated-images-into-canvas/

[4] J. J. Buchholz. (2014) Display symbolic expression in web browser. [Online]. Available: http://www.mathworks.de/matlabcentral/fileexchange/32694-display-symbolic-expression-in-web-browser

[5] M. Consortium. (2014) Mathjax. [Online]. Available: http://www.mathjax.org/