Erstellung einer Augmented-Reality-App für Android mit Unity und Vuforia

Projektbericht

im Sommersemester 2016
Abgabetermin: 18.07.2016

Von: Nicole Fabricius

E-Mail: fani1016@hs-karlsruhe.de
Matrikelnummer: 54302

Veranstaltung: Media Engineering
Dozent: Dipl.-Ing. Martin Schober

Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät: Informationsmanagement und Medien
Studiengang: Kommunikation und Medienmanagement (M.Sc.)

Inhaltsverzeichnis

1 Idee

In der Projektveranstaltung sollte eine Augmented-Reality-gestützte, mobile Hilfe-Anwendung für ein Android-Tablet in Verbindung mit einem Content-Delivery-Portal (CDP) konzipiert und entwickelt werden. Die praktische Umsetzung bestand somit aus 2 Teilen; der Programmierung des CDP mit HTML5-Methoden und der Umsetzung der AR-Elemente in Unity und Vuforia, wobei der Schwerpunkt auf dem AR-Teil liegen sollte.

Die AR-Elemente sollten zum einen eingesetzt werden, um dem Anwender die verschiedenen Komponenten des zu dokumentierenden Produkts auf dem Tablet-Display anzuzeigen, z. B. durch Hervorheben einer Komponente, wenn darauf getippt wird. Da in einer AR-Anwendung aus Usability-Gründen (wie Überladung mit Information) nicht sämtliche Informationen zu einem Produkt gegeben werden können, sollte neben der AR-App auch ein CDP erstellt werden, das weitere Informationen zum Produkt enthält. Bei Anwahl einer Komponente sollte der Einstieg in das CDP möglich sein, wobei die ausgewählte Komponente automatisch für die Facette „Komponente“ übernommen und alle Module zu dieser Komponente angezeigt werden sollten. Die Module sollten dann mithilfe der facettierten Suche weiter gefiltert werden können.

Zum anderen sollten die AR-Elemente bei anleitenden Modulen das Verständnis des Nutzers fördern und die durchzuführenden Schritte demonstrieren. Dies sollte mindestens durch einfache Pfeile erfolgen, die eine Bewegungsrichtung eines Hebels o.Ä. anzeigen. Idealerweise sollten jedoch 3D-Animationen eingesetzt werden, die die auszuführende Bewegung der Komponente zeigen.

Im CDP sollte neben einer facettierten Suche anhand der Modul-Metadaten eine Volltextsuche umgesetzt werden. Zusätzlich sollte die „traditionelle“ Navigation über das Inhaltsverzeichnis möglich sein. Die Klassifikation der Hilfe-Module, die für die facettierte Suche im CDP verwendet werden sollte, erfolgte mithilfe der PI-Klassifikation von Prof. Dr. Ziegler.

Um die PI-Klassifikation sinnvoll anwenden und den Nutzen einer facettierten Suche im CDP demonstrieren zu können, musste das zu dokumentierende Produkt komplex genug sein: Das Produkt sollte für eine sinnvolle Informationsklassifikation z. B. über verschiedene Phasen des Produktlebenszyklus anwendbar sein, und für eine sinnvolle Produktklassifikation z. B. über mehrere unterschiedliche zu dokumentierende Komponenten verfügen. Das Produkt sollte dabei jedoch nicht zu komplex sein, um Einsatzmöglichkeiten und Nutzen von AR in der Technischen Dokumentation an einem einfachen Beispiel demonstrieren zu können. Eine Kaffeemaschine erfüllt alle diese Kriterien, weshalb die AR-App für eine Kaffeemaschinen-Anleitung umgesetzt wurde.

Im Fazit dieses Projektberichts sollte außerdem kurz darauf eingegangen werden, welche zusätzlichen Möglichkeiten und Vorteile Augmented Reality in der Technischen Redaktion allgemein bietet, und welche Anpassungen der Dokumentation bei Umstellung auf AR-Methoden notwendig sind.

2 Vorgehensweise und verwendete Tools

2.1 Programmierung des Content-Delivery-Portals

2.1.1 Content in XML erfassen und klassifizieren

XML wurde als Erstellungsformat gewählt, da über das XSL-Stylesheet sowohl strukturelle Änderungen als auch Änderungen des Layouts flexibel und zentral für alle resultierenden HTML-Module angepasst werden konnten. Zudem konnte in XML das Informationsmodell PI-Mod verwendet werden, um alle Informationen zu erfassen, die später für die facettierte Suche im CDP verwendet werden sollten – entweder in Form von XML-Elementen (z.B. task, descriptive) oder in Form von Attributen (z.B. infoclass-1, prodclass-1). Darüber hinaus ist PI-Mod ein semantisches Informationsmodell, was für die folgende regelbasierte XSL-Transformation sinnvoll war, denn so konnte sichergestellt werden, dass jede Schrittanleitung, jeder Sicherheitshinweis, etc. konsistent transformiert und formatiert wird.

Die Anleitung zur ausgewählten Kaffeemaschine ist frei im Internet als PDF erhältlich. Aus dieser wurden die Inhalte in XML übertragen und teilweise bezüglich Formulierung und Strukturierung optimiert. Jedoch war die inhaltliche Optimierung kein Schwerpunkt dieser Projektarbeit. Bei der Auswahl der Inhalte für das CDP wurde darauf geachtet, eine möglichst große Bandbreite an Klassifikationen (d.h. mehrere Komponenten, Modultypen, etc.) für die facettierte Suche zu ermöglichen. Insgesamt wurden 25 Topics erfasst.

2.1.2 Content mit XSL transformieren

Anschließend wurden die Module mithilfe von XSLT nach HTML transformiert, um eine Anzeige im Browser zu ermöglichen. Auf die Funktionsweise von XSLT soll in diesem Projektbericht nicht weiter eingegangen werden.

2.1.3 Portal mit HTML erstellen

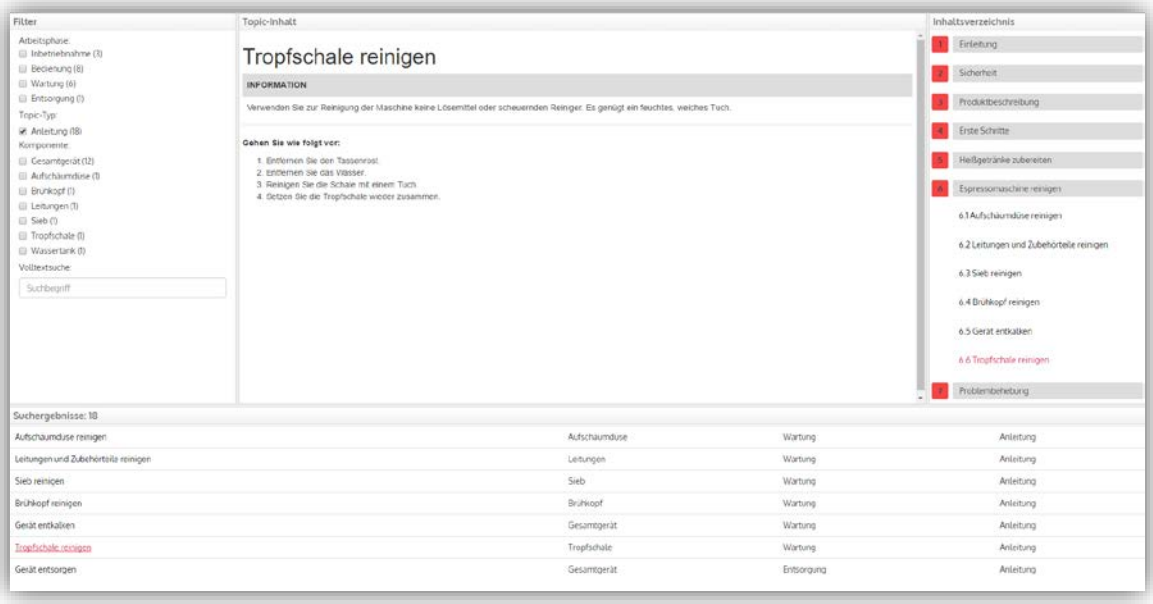

Abb. 1: Content-Delivery-Portal

An dieser Stelle wird der Aufbau des Portals in HTML erklärt. Die Programmierung der Suchfunktionen mit JavaScript wird in Kapitel 2.1.4 beschrieben. Die HTML-Datei für das Portal wurde aus der XML-Datei mit der Kaffeemaschinen-Anleitung mittels XSLT generiert.

2.1.3.1 Ein- und ausblendbare Viewlets mit EasyUI

Da auf einem Tablet nur begrenzter Platz zur Anzeige vorhanden ist, sollte der Anwender des CDP die Möglichkeit haben, Viewlets, die gerade nicht benötigt werden, auszublenden und später wieder einzublenden. Um auf schnelle und einfache Weise ein- und ausblendbare Viewlets zu erhalten, wurde auf das Framework EasyUI zugegriffen. Dafür wird mithilfe von div-Containern mit region-Attribut definiert, an welcher Stelle (z.B. *center*) der Content des div-Containers stehen soll. Diese div-Container werden im Browser zu ein- und ausblendbaren frame-Elementen umgewandelt.

```
<body>
      <div class="easyui-layout" style="width:100%; height:100%">
            <div region="west" split="true" title="Filter" style="width:20%;">
            …
            </div>
            <div id="content" class="target" region="center" title="Topic-Inhalt"
            style="padding:5px;">
            …
            </div>
            <div region="east" split="true" title="Inhaltsverzeichnis"
            style="width:20%;">
            …
            </div>
            <div id="south" region="south" split="true" title="Suchergebnisse"
            style="height:30%;">
            …
            </div>
      </div>
</body>
```

2.1.3.2 Suche-Viewlet (Facetten und Volltextsuche)

Auf der linken Seite im Portal wurden Facetten und Volltextsuche als Formular-Elemente (type=“checkbox“ und type=“text“) in HTML umgesetzt.

2.1.3.3 Content-Viewlet

Mittig im Portal wird der Content des aktuell ausgewählten Topics in einem object-Element angezeigt. Welches Topic angezeigt wird, wird über das data-Attribut bestimmt. Auf der Startseite wird ein Bild der Kaffeemaschine angezeigt.

```
<div id="content" class="target" region="center" title="Topic-Inhalt">
      <object style="width:100%;" data="images/Delonghi_Icona.jpg"/>
</div>
```

2.1.3.4 IHVZ-Viewlet

Auf der rechten Seite im Portal wurde das Inhaltsverzeichnis der Hilfe als Aufzählung (ordered list) umgesetzt.

2.1.3.5 Suchergebnis-Viewlet

Die Suchergebnisliste am unteren Rand wurde als Tabelle umgesetzt. Neben einem Link auf das Topic sind in der Tabelle die Metadaten des Topics enthalten. In der Suchergebnisliste werden beim Einstieg ins Portal alle 25 Topics angezeigt. Je nachdem, was der Anwender in das Suche-Viewlet eingibt, werden nicht zutreffende Topics über JavaScript aus der Tabelle ausgeblendet.

2.1.4 Portal programmieren mit jQuery

Die Funktionen für Suche und Navigation im CDP wurden mit dem JavaScript-Framework jQuery umgesetzt.

2.1.4.1 Navigation im Inhaltsverzeichnis

Beim Navigieren im Inhaltsverzeichnis werden bei Klick auf ein Kapitel erster Ebene (Link der Klasse *strong*) die darunterliegenden Topics im Inhaltsverzeichnis ein- und ausgeblendet:

```
$('a.strong').on('click',function(){
       if($(this).next().is('ol:hidden')){
                $(this).next('ol').show();
         }
         else{
               $(this).next('ol').hide();
         }
    });
```

2.1.4.2 Topic anzeigen

Links auf Topics besitzen sowohl im Inhaltsverzeichnis als auch in der Suchergebnisliste die Klasse *ihvz*. Bei Klick auf einen dieser Links wird das Topic über das data-Attribut des object-Elements im Portal angezeigt. Der Link auf das Topic wird sowohl im Inhaltsverzeichnis als auch in der Suchergebnisliste rot eingefärbt, damit der Anwender sehen kann, an welcher Stelle im Portal er sich befindet. Im Inhaltsverzeichnis wird zu der Stelle des Topics gesprungen und alle anderen Kapitel zugeklappt.

```
$('a.ihvz').on('click',function(){
        var ID = $(this).attr('id');

            $('a.ihvz').css("color", "rgb(0, 0, 0)");

            var Inhalt = $(this).text();

            //Bei Klick auf einen Link zu einem Topic in der Suchergebnisliste,
            färbe den Link sowohl im IHVZ als auch in der Suchergebnisliste rot
            $('a.ihvz:contains("'+Inhalt+'")').css("color", "rgb(220,20,60)");
            $(this).css("color", "rgb(220, 20, 60)");

            //Springe außerdem zur richtigen Stelle im IHVZ und klappe alle
            anderen Kapitel zu
            $('a.ihvz:contains("'+Inhalt+'")').parent('li').parent('ol').parent('
            li').siblings().children('ol').hide();
            $('a.ihvz:contains("'+Inhalt+'")').parents('ol').show();

            //Zeige das Topic im Object-Element an
            $("object").attr('data', ID + ".htm");
            $("object").attr('style', "width:100%;height:100%");
    });
```

2.1.4.3 Anzahl der vorhandenen Suchergebnisse pro Facette angeben

Um die Anzahl der vorhandenen Suchergebnisse für eine Facette zu ermitteln, wird überprüft, wie viele Suchergebnisse (Klasse *resultblock*) zu der Facette passen. Dies wird anhand der drei data-tag-Attribute ermittelt. Anschließend werden die Ergebnisse für alle 3 data-tag-Attribute addiert und, sofern passende Ergebnisse vorhanden sind, hinter den Facettenwert geschrieben. Wenn keine passenden Ergebnisse vorhanden sind, ist die Facette für die weitere Einschränkung der Suchergebnisse nicht relevant und wird deshalb ausgeblendet. So wird eine leere Ergebnismenge vermieden.

```
var category = $(this).val();
var categoryid = $(this).attr("id");

var len1 = $('tr:visible').filter(function() {
      return $(this).attr('data-tag-info1') == category;
      }).length;

var len2 = $('tr:visible').filter(function() {
      return $(this).attr('data-tag-info2') == category;
      }).length;

var len3 = $('tr:visible').filter(function() {
      return $(this).attr('data-tag-prod') == category;
      }).length;

var len = len1 + len2 + len3

//Wenn passende Ergebnisse vorhanden sind, schreibe diese in Klammern hinter den
Facettenwert
if (len1 + len2 + len3 != 0
      {
      $('label[for="'+categoryid+'"]').html($(this).val() + ' (' + len + ')' );
      }
//Wenn keine passenden Ergebnisse vorhanden sind, verstecke den Facettenwert
else
      {
      $('.checkbox label[for="'+categoryid+'"]').hide();
      $('.checkbox input[id="'+categoryid+'"]').hide();
      }
```

2.1.4.4 Reaktion auf ausgewählten Facettenwert

Für jede Facette (Komponente, Phase und Topic-Typ) wird ein Array erstellt. Angewählte Facettenwerte dieser Facette werden in den Array geschoben. Anschließend werden die angewählten Facettenwerte mit den data-tag-Attributen der Suchergebnisse (Klasse *resultblock*) verglichen. Gibt es eine Übereinstimmung, wird das Suchergebnis eingeblendet, wenn nicht, wird es ausgeblendet.

```
//Erstelle einen Array für jede Facette
var category_list1 = [];
…

//Schiebe angewählte Werte in den jeweiligen Array
$('.category1:checked').each(function(){
      var category1 = $(this).val();
      category_list1.push(category1);
  });
…

//Blende Zeilen aus der Suchergebnisliste (Klasse resultblock) aus oder ein, je
nachdem, ob ein passender Facettenwert ausgewählt wurde
//Blende Facettenwerte ein, wenn es ein passendes Ergebnis dazu gibt
$('.resultblock').each(function(){
      var item1 = $(this).attr('data-tag-info1');
      var item2 = $(this).attr('data-tag-prod');
      var item3 = $(this).attr('data-tag-info2');

      if(category_list1.length == 0 && category_list3.length == 0 &&
      category_list2.length != 0 && jQuery.inArray(item2,category_list2) > -1){
            $(this).fadeIn('slow');
            $('.category1[value="'+item1+'"],
            .category2[value="'+item2+'"],
            .category3[value="'+item3+'"]').fadeIn('slow');
            $('.category1[value="'+item1+'"],
```

```
            .category2[value="'+item2+'"],
            .category3[value="'+item3+'"]').next().fadeIn('slow');
             }
            else if(category_list1.length == 0 && category_list2.length == 0 &&
            category_list3.length != 0 && jQuery.inArray(item3,category_list3) >
            -1){
                  $(this).fadeIn('slow');
                  $('.category1[value="'+item1+'"],
                  .category2[value="'+item2+'"],
                  .category3[value="'+item3+'"]').fadeIn('slow');
                  $('.category1[value="'+item1+'"],
                  .category2[value="'+item2+'"],
                  .category3[value="'+item3+'"]').next().fadeIn('slow');
              }
      else{
            $(this).hide();
              }
        });
```

2.1.4.5 Volltextsuche

Bei Klick auf den Suche-Button wird der Inhalt des Suchfeldes extrahiert und mit dem Inhalt der Suchergebnisse (Klasse *resultblock*) verglichen. Wenn der Inhalt des Suchfeldes und des Suchergebnisses nicht zusammenpassen, wird das Suchergebnis ausgeblendet. Anschließend wird die Anzeige der Anzahl der gefundenen Suchergebnisse (Variable *count*) aktualisiert. Bei Klick auf den Zurücksetzen-Button wird das Suchfeld geleert und die Suche zurückgesetzt.

Die Volltextsuche schränkt die Suchergebnisse der facettierten Suche weiter ein, da sie nur die Tabellenzeilen durchsucht, die nach der facettierten Suche als Ergebnisse sichtbar sind. Daher sollte im Portal zuerst die facettierte Suche und danach die Volltextsuche zur Verfeinerung der Suche verwendet werden.

```
$('.searchbutton').on('click',function(){

       // Wert des Eingabefelds in Variable speichern, Ergebnis-Zahl auf 0 setzen
      var id = $(this).attr('id');
      var volltextsuche = $("#volltextsuche").val() , count = 0;

      // Wenn Zurück-Button geklickt, leere das Suchfeld und führe myFunction aus
      if(id == 'zurueck'){
            $("#volltextsuche").val('');
            myFunction();
      }
      //Wenn Suche-Button geklickt
      else{
        // Durchsuche vorhandene Suchergebnisse (=sichtbare Tabellenzeilen)
        $("tr:visible td:first-child").each(function(){
            // Blende unpassende Zeilen aus
            if ($(this).text().search(new RegExp(volltextsuche, "i")) < 0) {
                $(this).parent().fadeOut();
            // Blende passende Zeilen ein und erhöhe die Ergebnis-Zahl
            } else {
                $(this).parent().show();
                count++;
            }
        });

        // Schreibe die Ergebnis-Zahl in den Panel-Titel
        var numberItems = count;
        $('.panel-title:last').html("Suchergebnisse: "+count);
            }
   });
```

2.1.4.6 Gefundene Suchergebnisse zählen und anzeigen

Die gefundenen Suchergebnisse, d.h. die aktuell sichtbaren Tabellenzeilen, werden gezählt und in den Panel-Titel der Suchergebnisliste geschrieben.

```
var cnt = $('tr:not([style*="display: none"])').length - 1;
$( '.panel-title:last' ).html("Suchergebnisse: " + cnt);
```

2.1.4.7 In AR-App angewählte Komponente vorauswählen

Dazu wurde im C#-Skript mit der Verlinkung auf das Portal in Unity (s. Kapitel 2.2.4.5) je nach angewählter Komponente ein anderer Zusatz nach dem Muster *?facet_id* an die URL des Portals angehängt. In der JavaScript-Datei wird dieser Zusatz mit

```
var query = window.location.search.substring(1);
```

ausgelesen und mit dem #-Zeichen zur ID der Facette in HTML erweitert:

```
var checkid = "#" + query;
```

Anschließend wird die entsprechende Facette mithilfe der ID angewählt:

```
$(checkid).prop('checked', true);
```

2.2 Erstellung der AR-App in Unity mit der Vuforia-Erweiterung

2.2.1 Einführung

2.2.1.1 Was ist Unity?

Unity ist eine Laufzeit- und Entwicklungsumgebung für Computerspiele und andere interaktive 3D-Anwendungen des Unternehmens Unity Technologies. Zielplattformen von Unity sind Computer, Spielkonsolen, mobile Geräte und Webbrowser.

Zu Beginn der Projektarbeit wurden mehrere Entwicklungsumgebungen oberflächlich getestet. Unity in Verbindung mit Vuforia hat den Vergleich für sich entschieden, weil Unity seit Jahren eine weit verbreitete 3D-Entwicklungsumgebung ist und umfangreiche Funktionen bietet, die mithilfe der Vuforia-Erweiterung auch für AR-Anwendungen verwendet werden können. Dementsprechend gibt es zu Unity sehr umfangreiches Informationsmaterial.

2.2.1.2 Was ist Vuforia?

Vuforia ist ein Software Development Kit (SDK) für die Entwicklung von Augmented-Reality-Anwendungen, das 2015 von PTC gekauft wurde. Als Erweiterung der Unity Game Engine unterstützt Vuforia sowohl die native Entwicklung für iOS und Android als auch die Erstellung von Anwendungen im Unity Editor. Letzteres wurde in diesem Projekt angewendet.

Derzeit läuft Vuforia unter Windows offiziell nur in der 32-Bit-Version von Unity (vgl. PTC Inc. 2016). Hier ist jedoch anzumerken, dass zu Projektbeginn versehentlich mit der 64-Bit-Version gearbeitet wurde, und auch diese weitgehend fehlerfrei zu funktionieren scheint. Jedoch kann die Anwendung hier nicht getestet werden, da Laptop-Kameras in Unity nicht erkannt werden und ein Fehler anzeigt wird. Die App zu packen und auf das Endgerät zu laden, funktioniert jedoch.

2.2.1.3 Grundlegendes zu C#

Grundsätzlich kann in Unity in C#, JavaScript oder Boo programmiert werden. Unity-Anfängern wird jedoch C# empfohlen, da viele der aktuellen Tutorials und Beispiele im Internet auf C#

basieren. Auch die offiziellen Unity-Tutorials verwenden C# (jedoch wird der Code zusätzlich als JavaScript bereitgestellt). Aus diesem Grund wurde auch im vorliegenden Projekt C# verwendet.

C# ist eine Programmiersprache für .NET, die von Microsoft in Zusammenarbeit mit Anders Heijlsberg, dem Erfinder von Delphi, speziell für diese Plattform entwickelt wurde mit dem Ziel, eine „modern[e], durchgängig objekt- wie auch komponentenorientiert[e], und vor allem verständlich[e]“ (Roden 2010) Sprache zu schaffen.

Ein C#-Skript besteht aus einer oder mehreren Klassen, in denen der Code strukturiert wird. Jeder ausführbare C#-Code muss in einer solchen Klasse enthalten sein, wobei der Name der Klasse (hier *MyScript*) der Benennung des Skripts im Unity-Projektverzeichnis entsprechen muss. Noch vor der ersten Klasse stehen using-Direktiven, die die Verwendung von Typen in einem Namespace ermöglichen, sodass die Verwendung eines Typs in diesem Namespace nicht qualifiziert werden muss (vgl. Microsoft 2016a). Innerhalb der Klassen werden Variablen (mit Angabe des Typs, hier *int*) und Funktionen (sogenannte „Methoden“) definiert.

```
using UnityEngine;
using System.Collections;

public class BeispielSkript : MonoBehaviour
      {
            private int MyVar = 2016;

            public void MyFunction ()
            {
            Debug.Log(“Hello World im Jahr” + MyVar);
            }
      }
```

Abb. 2: Struktur eines C#-Skripts

Für Variablen, Klassen und Funktionen werden Zugriffsebenen festgelegt. Die beiden wichtigsten Zugriffsebenen sind public und private:

- „public: Auf den Typ oder Member kann von jedem Code in der gleichen Assembly oder einer anderen Assembly, die darauf verweist, zugegriffen werden.
- private: Auf den Typ oder Member kann nur von Code in der gleichen Klasse oder Struktur zugegriffen werden.“ (Microsoft 2016b)

Bei Funktionen muss zusätzlich, je nachdem, was eine Funktion zurückgibt, ein anderes Schlüsselwort (z.B. „int“) vor den Namen der Funktion gesetzt werden. Das Schlüsselwort „void“ bedeutet, dass die Funktion nichts zurückgibt. Die in Abb. 2 verwendete Funktion gibt lediglich den Satz „Hello World im Jahr 2016“ in der Konsole aus.

2.2.2 Aufbau und Arbeitsweise von Unity

2.2.2.1 Viewlets

Die wichtigsten Viewlets in Unity sind die Szene, die Objekthierarchie, das Projektverzeichnis und der Inspektor (s. Abb. 3). In der Szene wird die zu erstellende Anwendung, hier die AR-App, bearbeitet. Mithilfe der Icons oben links kann der Anwender in der Szene navigieren und z. B. ein Objekt in der Szene drehen, verschieben oder skalieren. Über den Play-Button oben in der Mitte kann die Anwendung direkt in Unity gestartet und somit getestet werden.

In der Objekthierarchie werden die Objekte, die in der Szene verwendet werden, verwaltet. Diese Objekte werden aus dem Projektverzeichnis per Drag & Drop in die Objekthierarchie gezogen.

Die Objekte können beliebig gruppiert und ineinander verschachtelt werden, um z. B. eine Animation für eine gesamte Objektgruppe (z. B. Siebhalter der Kaffeemaschine mit allen untergeordneten Komponenten) zu erstellen anstatt für einzelne Objekte.

Das Projektverzeichnis enthält sämtliche C#-Skripte, 3D-Modelle, Animationen, Materialien usw. sowie vorgefertigte Elemente, die im Projekt verwendet werden können. Der Inspektor zeigt Details zum in der Objekthierarchie oder im Projektverzeichnis ausgewählten Objekt an, wie Abmessungen oder Komponenten (z. B. C#-Skripte, Animationen, Renderer, Text).

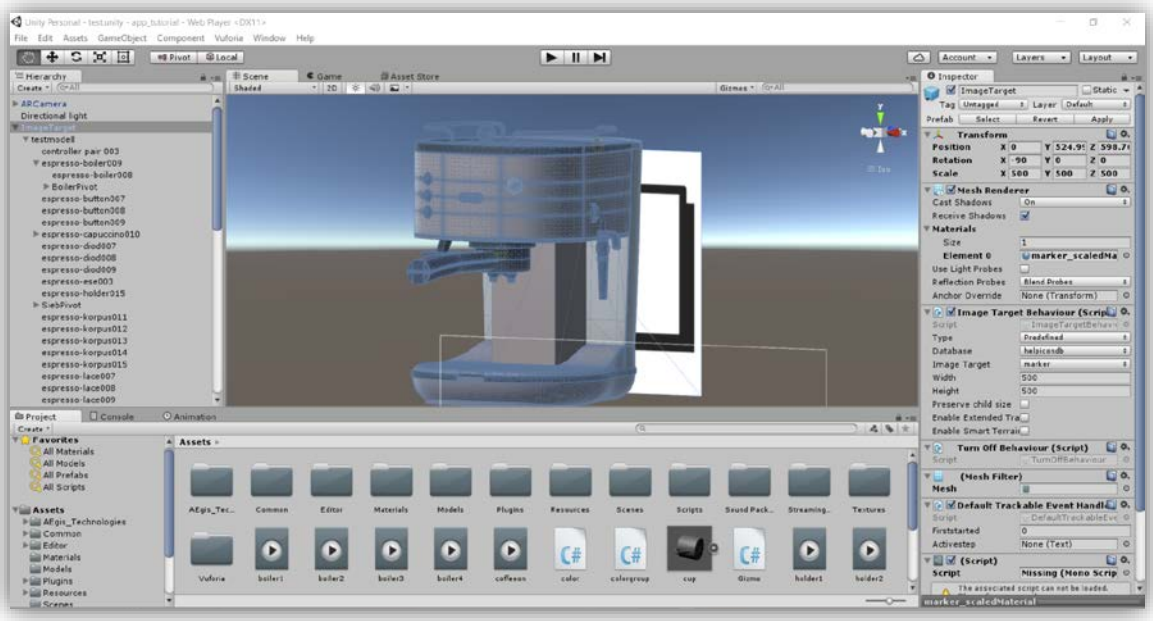

Abb. 3: Objekthierarchie (links), Szene (mittig), Inspektor (rechts), Projektverzeichnis (unten)

2.2.2.2 Objekte und ihre Komponenten

Alle Objekte in Unity haben je nach Objekttyp verschiedene voreingestellte Komponenten, die im Inspektor eingesehen werden können. Objekte eines 3D-Modells können z. B. einen Mesh Renderer oder einen Mesh Filter als Komponenten haben. Weitere Komponenten wie Animationen oder C#-Skripte, die für ein Objekt gelten sollen, können per Drag & Drop oder über das Kontextmenü hinzugefügt werden.

2.2.2.3 Verwendung von C#-Skripten

C#-Skripte, die in einer Anwendung verwendet werden sollen, müssen eine Komponente eines Objekts in der Objekthierarchie sein. Durch Drag & Drop des C#-Skripts auf das Objekt, auf das sich das Skript beziehen soll, wird das Skript als Komponente des Objekts hinzugefügt. Gibt es im Skript Variablen, für die noch kein Wert definiert wurde, muss dieser noch festgelegt werden (s. Kapitel 2.2.2.4).

Buttons sind hier die Ausnahme, da es für Buttons bereits eine Komponente *Button (Script) – On Click()* gibt, die für die Definition der Funktion des Buttons vorgesehen ist. In diesem Fall fügt man das C#-Skript mit der Funktion, die dem Button zugeordnet werden soll, einem Objekt auf der obersten Ebene der Objekthierarchie hinzu. Im Projekt wurde dafür ein leeres Objekt *MeineUI* angelegt, das alle C#-Skripte mit Funktionen für Buttons enthält. Das Objekt *MeineUI* muss dann per Drag & Drop der Komponente *Button (Script) – On Click()* eines Buttons zugewiesen und das gewünschte Skript sowie die gewünschte Funktion ausgewählt werden.

In C#-Skripten können außerdem Funktionen aus anderen Skripten (hier das Skript „schrittanl“ mit der Funktion „increase_counter“) wie folgt aufgerufen werden:

```
GameObject.Find("MeineUI").GetComponent<schrittanl>().increase_counter();
```

2.2.2.4 Verwendung von Variablen

Variablen müssen zu Beginn des C#-Skripts inklusive Angabe des Typs definiert werden:

```
private gameObject Gameobj;
```

Für die Zuweisung eines Werts der Variable gibt es 2 Möglichkeiten. Die Variable kann direkt im C#-Skript definiert werden:

```
Gameobj = GameObject.Find("testmodell");
```

Variablen aus einem anderen Skript (hier das Skript „schrittanl“ mit der Variable „mycounter“) können wie folgt verwendet werden:

```
mycounter2 = GameObject.Find("MeineUI").GetComponent<schrittanl>().mycounter;
```

Alternativ kann die Variable im Inspektor des Objekts definiert werden, an das das C#-Skript gehängt wurde. Dazu wird per Drag & Drop das Objekt, das als Wert der Variable verwendet werden soll, auf das dafür vorgesehene Feld gezogen. In Abb. 4 wird der Variablen *Gameobj* das Objekt *testmodell* zugewiesen, was das gleiche Ergebnis erzielt wie die oben stehende Zeile Code. Dieses Vorgehen hat den Vorteil, dass dasselbe Skript für mehrere Komponenten wiederverwendet werden kann und den Variablen des Skripts schnell und einfach je nach Bezugsobjekt ein anderer Wert zugewiesen werden kann.

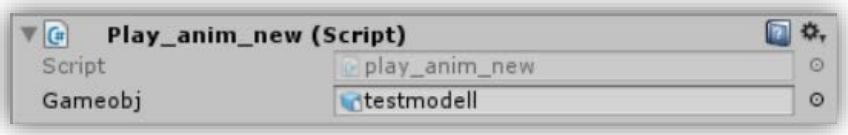

Abb. 4: Zuweisung einer Variablen über den Inspektor

2.2.3 Endergebnis: Funktionen der fertigen AR-App

In diesem Abschnitt soll ein Überblick über die Funktionen der fertigen App gegeben werden. In Kapitel 2.2.4 wird beschrieben, wie die Funktionen jeweils umgesetzt wurden. Die vollständigen Skripte, die für die App verwendet wurden, sind im Anhang aufgeführt.

Beim Öffnen der App muss die Kamera auf den Marker, einen 20€-Schein, gehalten werden. Bei Erkennen des Markers wird das 3D-Modell der Kaffeemaschine eingeblendet und eine Übersicht über die wichtigsten Tätigkeiten an der Kaffeemaschine angezeigt. (Anmerkung: Für das vorliegende Projekt wurde nur die Anleitung „Kaffee zubereiten“ umgesetzt.) Bei Klick auf „Zum Hilfe-Portal“ wird das CDP im Browser geöffnet. Wird auf eine der Anleitungen in der Schrittübersicht geklickt, wird die Schrittübersicht ausgeblendet und der erste Handlungsschritt der Anleitung sowie eine passende Animation dazu angezeigt. Durch Klicken auf den „Weiter“-Button wird jeweils zum nächsten Schritt mit passender Animation gewechselt. Zusätzlich zu den Animationen wird das 3D-Modell passend manipuliert und Objekte hinzugefügt (z. B. Kaffeetasse), ausgeblendet oder gefärbt (z. B. LED-Anzeigen).

Durch Klick auf den „Zurück“-Button hat der Anwender die Möglichkeit, sich den vorherigen Schritt noch einmal anzusehen. Soll z. B. Schritt 3 wiederholt werden, wird die Animation von Schritt 4 zurückgesetzt und die Animation von Schritt 3 wiederholt. Das Zurücksetzen der Animation von

Schritt 4 ist notwendig, damit das 3D-Modell die richtige Ausgangsposition für das Abspielen von Schritt 3 hat.

Wenn der Anwender sich in der Schrittanleitung befindet, kann er eine Komponente der Kaffeemaschine durch Berühren rot hervorheben. (Anmerkung: Für das vorliegende Projekt wurde dies nur für die Komponenten Wassertank, Milchaufschäumdüse und Siebhalter umgesetzt.) Gleichzeitig wird ein Textfeld mit einer Kurzbeschreibung der Komponente eingeblendet, sowie ein Button, über das CDP im Browser geöffnet wird, falls der Anwender weitere Informationen zur Komponente benötigt. In der facettierten Suche im CDP ist die berührte Komponente bereits vorausgewählt, sodass in der Suchergebnisliste nur Topics angezeigt werden, die sich auf die berührte Komponente beziehen. Bei Bedarf kann der Anwender die Suchergebnisse durch Verwendung der anderen Facetten oder der Volltextsuche weiter einschränken.

Durch erneutes Berühren der Komponente (oder Berühren einer der anderen Komponenten) werden Textfeld und Button wieder ausgeblendet und die Komponente in ihre ursprüngliche Farbe zurück gefärbt.

Durch den Schließen-Button oben rechts wird die Schrittanleitung geschlossen und die Schrittübersicht wieder angezeigt. Das 3D-Modell bleibt dabei in der Position des letzten Schrittes, damit bei erneutem Klick auf „Kaffee zubereiten“ aus der Schrittübersicht heraus die Anleitung beim zuletzt aktiven Schritt fortgesetzt werden kann.

Abb. 5: Einstieg in die App mit Schrittübersicht

Abb. 6: Schrittanleitung mit passender Animation

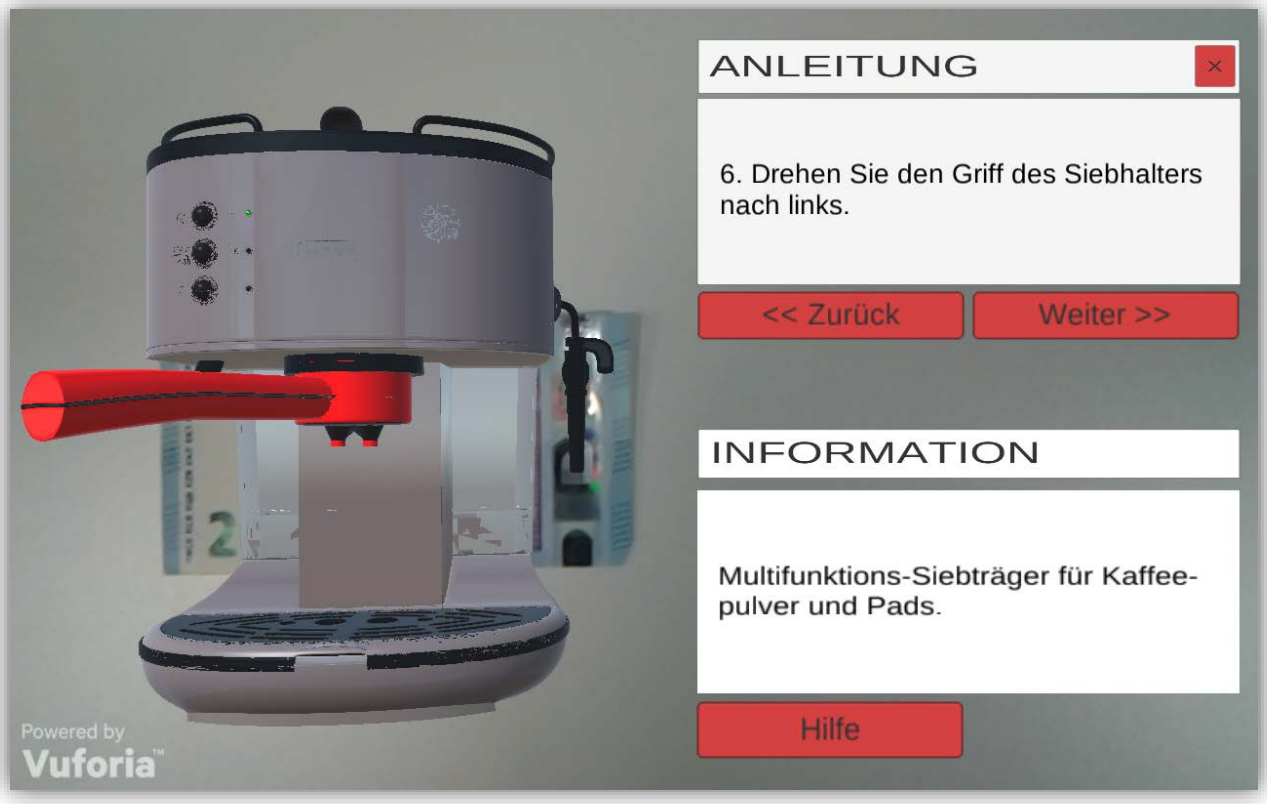

Abb. 7: Eingefärbte Komponente mit Kurzbeschreibung

2.2.4 Schritt für Schritt zur AR-App: Vorgehen und Erfahrungen

2.2.4.1 Projekt anlegen und einrichten

Nach dem Anlegen eines neuen Unity-Projekts muss die Vuforia-Erweiterung in Unity eingebunden werden, indem die .unitypackage-Datei vom Vuforia Developer Portal heruntergeladen und in Unity importiert wird. Dadurch werden spezielle Objekte für Vuforia in das Projektverzeichnis geladen. Im vorliegenden Projekt wurden davon die Objekte ARCamera und ImageTarget verwendet.

2.2.4.2 Bild- oder Objekterkennung einrichten

In Unity gibt es folgende Möglichkeiten, Bilder oder 3D-Objekte zu tracken (s. Abb. 8):

- Single Image: Beliebiges 2D-Bild
- Cuboid: Quaderförmiges 3D-Objekt (im Projekt nicht getestet)
- Cylinder: Zylindrisches 3D-Objekt (im Projekt nicht getestet)
- 3D Object: Selbst gescanntes 3D-Objekt

Die Targets werden im Target Manager im Vuforia Developer Portal verwaltet. Beim Anlegen eines Targets gibt man bei Cuboid und Cylinder jeweils die gewünschten Abmessungen (LxBxH) des Objekts an, bei Single Image und 3D Object lädt man eine Datei hoch.

Das Tracking hat mit allen im Projekt verwendeten ImageTargets funktioniert, wobei das Tracking besser funktioniert, je mehr Anhaltspunkte für das Tracking ermittelt werden können, d. h. je komplexer das Bild ist. Die Anhaltspunkte für das Tracking können ebenfalls im Target Manager eingesehen werden (s. Abb. 9).

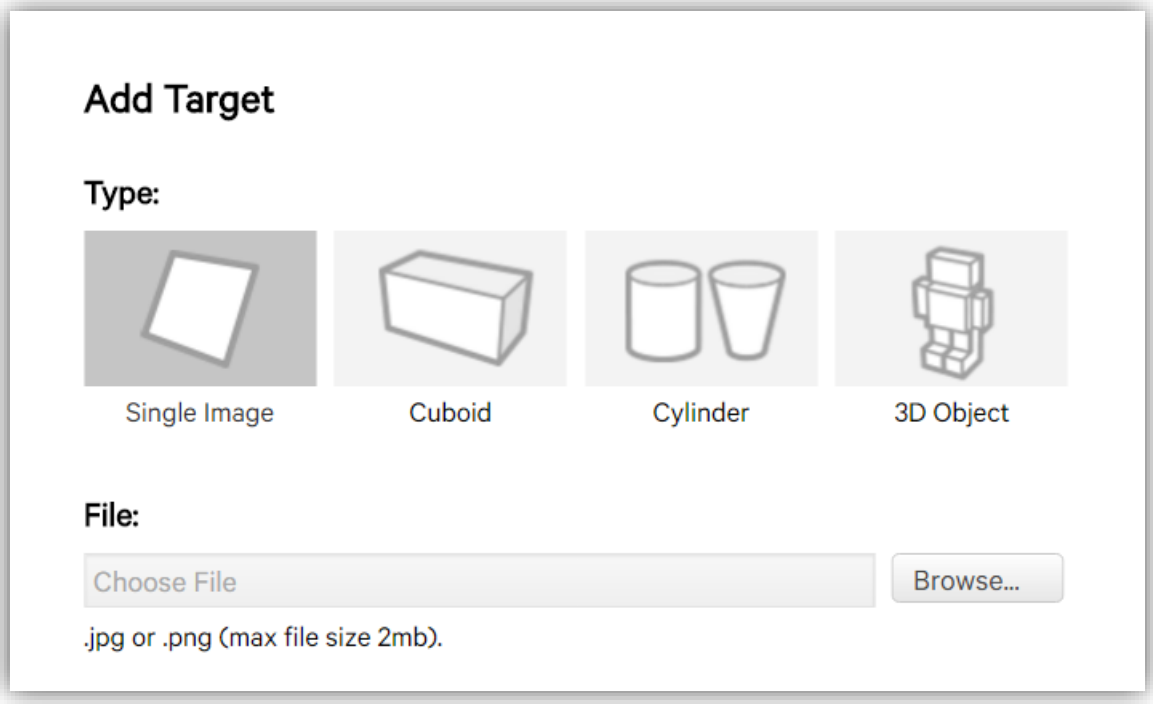

Abb. 8: Neues Target anlegen

Abb. 9: Gutes ImageTarget mit vielen Anhaltspunkten für das Tracking

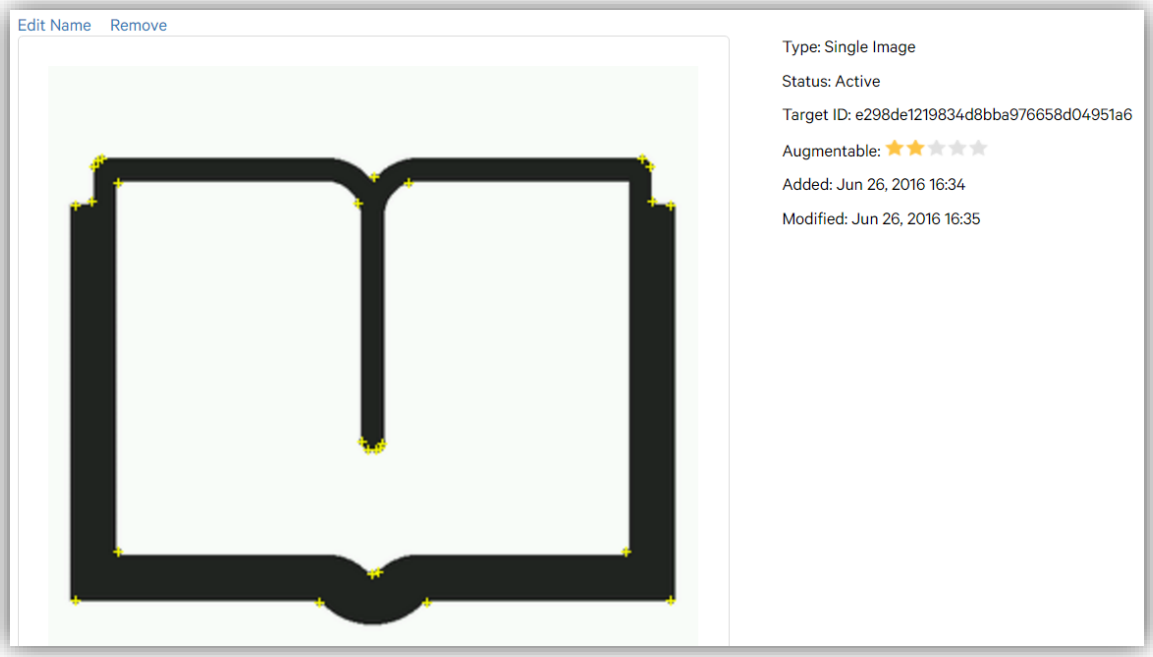

Abb. 10: Schlechtes ImageTarget mit wenigen Anhaltspunkten für das Tracking

Als 3D Object können nur Daten des Vuforia Object Scanner verwendet werden. Der Object Scanner ist eine App von Vuforia, die auf einem Smartphone oder Tablet installiert wird. Mithilfe dieser App kann ein 3D-Objekt gescannt werden, indem man das Smartphone oder Tablet einmal um das gesamte Objekt herumführt. Beim Scannen werden Anhaltspunkte im dreidimensionalen Raum erkannt und zu einem wiedererkennbaren 3D-Objekt zusammengesetzt.

Für ein erfolgreiches Scannen sind jedoch sehr gute Lichtverhältnisse (z. B. weißer Tisch vor einer weißen Wand, gut ausgeleuchtet) notwendig, da sonst nicht genug Anhaltspunkte erkannt werden. Zum Scannen ist außerdem ein von Vuforia bereitgestelltes „Object Scanning Target“ (s. Abb. 11) erforderlich, das im A4-Format ausgedruckt werden muss und zur Positionierung des

3D-Modells verwendet wird. Deshalb ist diese Methode nur zum Scannen von Objekten bis zu einer bestimmten Größe geeignet. Die im Projekt verwendete Kaffeemaschine ist vermutlich schon zu groß, da das Scannen mit dem Object Scanner nicht funktioniert hat und nur sehr wenige Anhaltspunkte erkannt wurden. Deshalb (und wegen der einfacheren Handhabung und Austauschbarkeit) wurde für das Projekt ein ImageTarget verwendet.

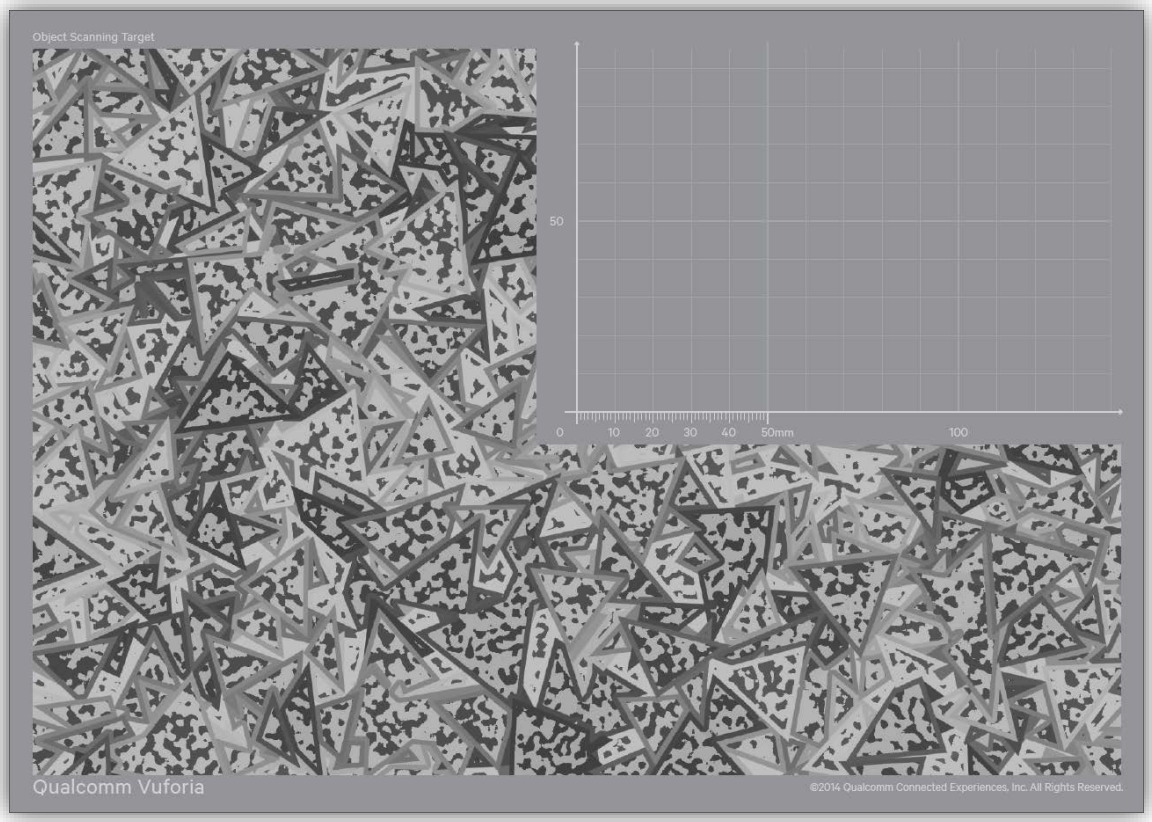

Abb. 11: Object Scanning Target. Das zu scannende Objekt wird im oberen rechten Viertel platziert.

Nachdem sie im Target Manager hinzugefügt wurden, können die ImageTargets heruntergeladen, in das Unity-Projektverzeichnis importiert und im ImageTarget-Objekt verwendet werden. Dazu wird dem ImageTarget-Objekt in der Objekthierarchie die heruntergeladene ImageTarget-Datei im Inspektor zugewiesen. Zusätzlich muss die ImageTarget-Datenbank im Inspektor des ARCamera-Objekts geladen werden.

2.2.4.3 3D-Modelle importieren

In der zu entwickelnden AR-App sollte das 3D-Modell der Kaffeemaschine angezeigt werden. Das 3D-Modell wurde auf 3Dsky.org im .fbx-Format erworben und konnte direkt in Unity importiert werden. Darüber hinaus kann Unity die Formate .dae, .3ds, .dxf und .obj lesen. Alle Objekte, die in der App bei Erkennen des Targets anzeigt werden sollen (mit Ausnahme von UI-Elementen), müssen in der Objekthierarchie ein Kind von ImageTarget sein – in diesem Projekt also das gesamte Kaffeemaschinen-Modell.

2.2.4.4 Benutzeroberfläche der AR-App gestalten

Unity stellt zur Gestaltung der Benutzeroberfläche verschiedene UI-Objekte zur Verfügung, z. B. Button, Text, Panel, Image, Scrollbar und Inputfield. Diese können einfach über die Menüleiste in die Szene eingefügt werden. Dabei wird, falls noch nicht vorhanden, automatisch ein Canvas-

Objekt erstellt, das allen UI-Objekten übergeordnet sein muss. Die UI-Objekte können, wie alle anderen Objekte, in der Szene skaliert, rotiert und positioniert werden.

Objekte wie Button oder Panel können außerdem mit Text versehen werden, wobei für den Text ein eigenes Objekt in der Objekthierarchie angelegt wird. Die Formatierung des Textes ist jedoch nur sehr eingeschränkt möglich. Zwar können Schriftart, Schriftstil und Schriftgröße im Inspektor eingestellt werden, aber nur für das ganze Text-Objekt und nicht für einzelne Zeichen. Es wurde auch kein Weg gefunden, Textformate in Unity zu speichern und zu verwalten, wie Formatvorlagen in Word oder attribute-sets in XSL. Problematisch ist außerdem, dass die Schrift mitskaliert und damit verzerrt wird, wenn beispielsweise ein Button nach Hinzufügen der Schrift größer skaliert wird.

Für das Canvas-Element sind verschiedene Bezugsräume einstellbar, wie Screen Space – Overlay, Screen Space – Camera und World Space. Von dieser Einstellung hängt u. a. ab, ob die UI-Objekte sich in der fertigen AR-App bei Bewegung der Kamera mit dem Kaffeemaschinen-Modell mitbewegen oder immer an derselben Stelle auf dem Display angezeigt werden. Letzteres wurde für die geplante App als nutzerfreundlicher empfunden und deshalb die Einstellung Screen Space – Overlay gewählt. In den Einstellungen für Canvas wird außerdem die Ziel-Displayauflösung (z. B. 1280 x 800 für das Samsung Galaxy Tab) und das Verhalten der UI-Elemente bei abweichender Bildschirmgröße festlegt (z. B. Scale With Screen Size – Match Width or Height). Die Default-Orientierung (Landscape, Portrait, Auto Rotation) sowie weitere Orientierungen, die möglich sein sollen, werden dagegen in den Einstellungen für die Android-Ausgabe (s. auch Kapitel 2.2.4.10) festgelegt.

Das responsive Verhalten der UI-Objekte wird nicht, wie aus der Web-Programmierung gewohnt, über prozentuale Größenangaben gesteuert, sondern über spezielle Anker (s. Abb. 12-Abb. 14). Das sind kleine Pfeile, die festlegen, an welchen Punkten sich die Größe eines Objekts orientiert. Dies kann je nach Anwendungsfall eine Ecke des Displays sein, oder auch alle vier Ecken eines Buttons. Da man die Anker für jedes einzelne UI-Objekt manuell und nach Augenmaß direkt in der Szene setzen muss, erscheint dieses Vorgehen unnötig kompliziert und unhandlich. Es wäre einfacher, im Inspektor prozentuale Abmessungen oder Abstände einzugeben.

Abb. 12: Buttons mit Anchors

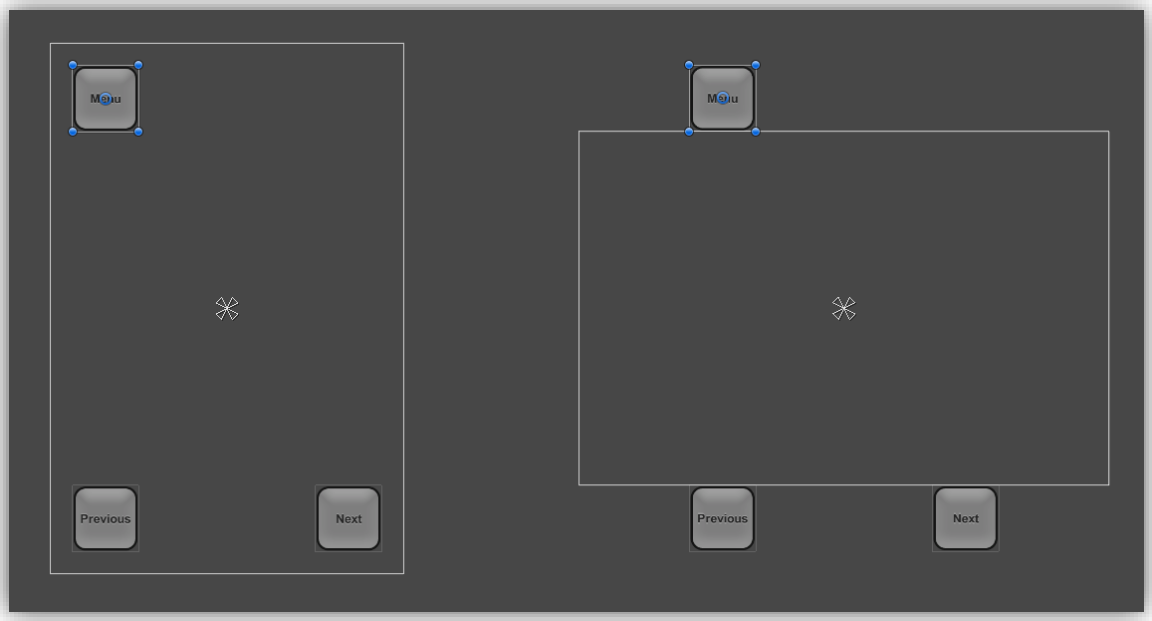

Abb. 13: Anker in der Mitte des Displays mit unerwünschtem Ergebnis (Unity Technologies 2016)

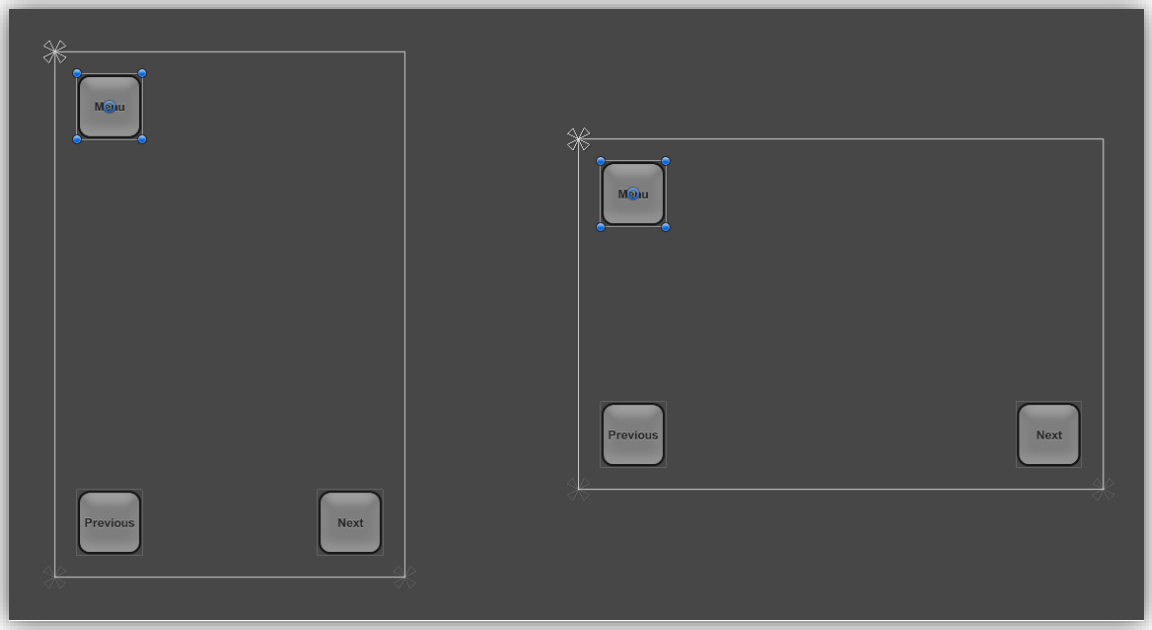

Abb. 14: Anker in einer Ecke des Displays mit gewünschtem Ergebnis (Unity Technologies 2016)

Um Buttons Funktionen (z. B. URL öffnen, 3D-Modell manipulieren) zuzuweisen, ist die Programmierung der Funktionen in C# notwendig (s. Kapitel 2.2.2.3).

Neben normalen Buttons sind über Vuforia außerdem Virtual Buttons möglich. Diese Buttons sind echte AR-Objekte, d. h. sie werden nicht durch Berühren des Buttons auf dem Display gesteuert, sondern durch Berühren des Buttons in der realen Welt (s. Abb. 15). Dies ist vor allem für Anwendungen für AR-Brillen sehr interessant. Diese Art von Button wurde im Projekt ausprobiert, aber nicht im Endergebnis verwendet, weil die Navigation über normale Buttons für die geplante App als nutzerfreundlicher empfunden wurde.

Abb. 15: Interaktion mit Virtual Buttons (YouTube 2015)

2.2.4.5 Über Buttons auf externe Webseiten verlinken

URLs werden über die C#-Funktion Application.OpenURL im Browser geöffnet. Im Projekt wurde diese Funktion verwendet, um nach Berühren einer Komponente auf dem Bildschirm über einen Button in das CDP mit Topics zu dieser Komponente springen zu können. Damit je nach gewählter Komponente ein anderer Wert für die Facette „Komponente“ in der Suche im CDP angewählt wird, wurde die URL des CDP außerdem je nach angewählter Komponente mit einem anderen Zusatz nach dem Muster *?facet_id* erweitert:

```
Application.OpenURL("http://www.technischeredaktion.com/icona/out/portal.htm?check
18");
```

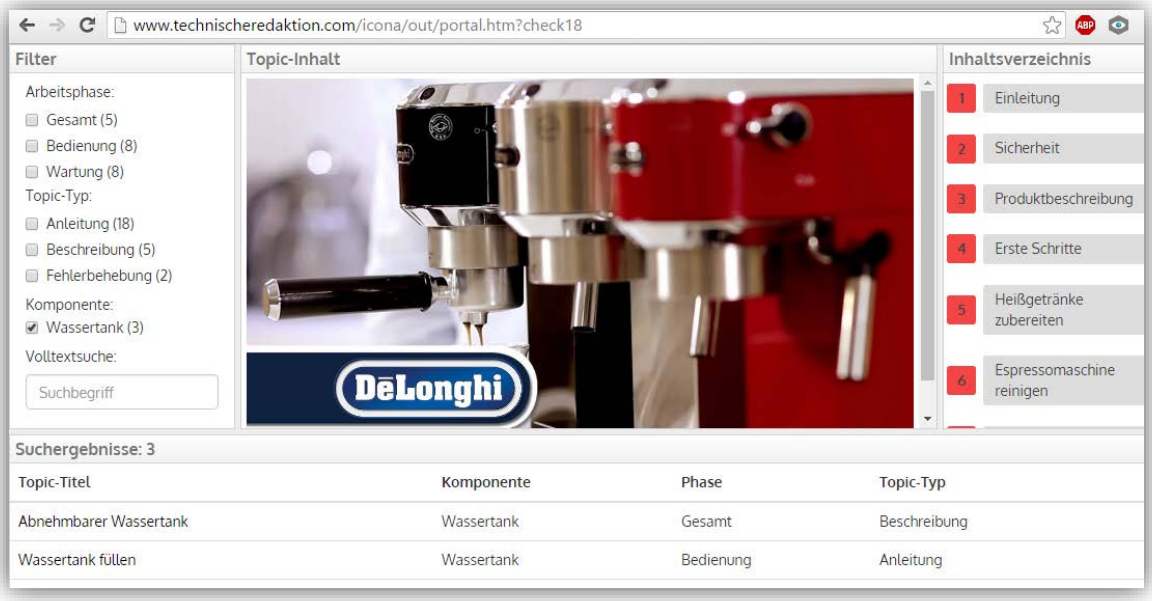

Abb. 16: Über URL-Zusatz vorausgewählte Facette "Wassertank"

In der JavaScript-Datei für das CDP wird der Zusatz ausgewertet (s. Kapitel 2.1.4.7) und je nach Inhalt des Zusatzes ein anderer Wert für die Facette „Komponente“ aktiv gesetzt (s. Abb. 16).

Wie in Kapitel 2.2.2.3 beschrieben, muss das C#-Skript mit der Funktion, die ein Button ausführen soll, einem Objekt auf der obersten Ebene der Objekthierarchie als Komponente hinzugefügt und anschließend der Komponente *Button (Script) – On Click()* des Buttons zugewiesen werden.

2.2.4.6 UI-Objekte ein- und ausblenden

Allgemein

Für Wassertank, Siebhalter und Milchaufschäumdüse wurden je eine Textbox mit einer Kurzbeschreibung und ein Button zum CDP erstellt, die immer an der gleichen Stelle auf dem Display erscheinen sollten, wenn die jeweilige Komponente auf dem Display berührt wird. Deshalb mussten die Objekte überlagert werden. Je nach berührter Komponente mussten also die zugehörigen beiden UI-Objekte eingeblendet und alle anderen UI-Objekte ausgeblendet werden.

Einzelne UI-Objekte können über die Funktion SetActive(false) bzw. SetActive(true) versteckt und wieder angezeigt werden:

```
GameObject.Find("button1").SetActive(false);
GameObject.Find("button2").SetActive(true);
```

Ist der Wert false, wird das Element versteckt; ist der Wert true, wird es wieder eingeblendet.

Sonderfall: Ein- oder Ausblenden beim (ersten) Erkennen des Markers

Das Verhalten der App bei Erkennen des Markers wird über die Funktion OnTrackerFound() im mitgelieferten Skript DefaultTrackableEventHandler.cs, das eine Komponente des ImageTarget-Objekts ist, gesteuert.

Da beim Erkennen des Markers eine Übersicht („Panel2“) über die wichtigsten Schritte zur Kaffeemaschine eingeblendet und die Schrittanleitungen („Panel1“) ausgeblendet werden sollten, wurde dieses Skript um folgende Zeilen erweitert:

```
if(firststarted==0)
	{
	Debug.Log("Allererster Start!");
	//Schrittanleitung ausblenden
	GameObject.Find("Panel").GetComponent<CanvasGroup>().alpha = 0;

	GameObject.Find("Panel").GetComponent<CanvasGroup>().interactable = false;
	//Schrittübersicht einblenden
	GameObject.Find("Panel2").GetComponent<CanvasGroup>().alpha = 1;

	GameObject.Find("Panel2").GetComponent<CanvasGroup>().interactable = true;

	//Variable firststarted auf 1 setzen, damit oben stehender Code nur beim
	ersten Erkennen ausgeführt wird
	firststarted=1;
	}
```

Da den Panel-Objekten jeweils weitere Objekte wie Buttons und Textboxen untergeordnet waren, konnte hier nicht wie oben die Funktion SetActive verwendet werden, die nur auf einzelne Objekte und nicht auf Objektgruppen anwendbar ist. Deshalb wurde die Sichtbarkeit der Panel-Objekte über alpha gesteuert: Wenn alpha = 0 ist, wird das Objekt ausgeblendet; bei alpha = 1 eingeblendet. Da Objekte aber bei alpha = 0 trotzdem noch auf Touch-Events reagieren, muss zusätzlich die Interaktivität ausgeschaltet werden.

Die Übersicht über die wichtigsten Schritte sollte nur beim ersten Erkennen des Markers eingeblendet werden und nicht jedes Mal, wenn der Marker bei Verwendung der App verloren geht und wieder gefunden wird. Deshalb wurde eine Variable firststarted verwendet. Firststarted ist beim Starten der App 0 und wird nach dem ersten Erkennen des Markers auf 1 gesetzt, weshalb die if-Schleife nur 1 Mal ausgeführt wird und danach nicht mehr.

2.2.4.7 Teile des 3D-Modells bei Berührung einfärben

In Unity gibt es eine vordefinierte OnMouseDown()-Funktion, die aktiviert wird, sobald ein Klick oder ein Touch-Event registriert wird. Diese Funktion wurde im Projekt dazu verwendet, bestimmte Aktionen auszuführen, wenn in der AR-App eine Komponente der Kaffeemaschine berührt wird. Dabei sollte Folgendes passieren:

- Teile des 3D-Modells einfärben
- Kurzbeschreibungstext und Button mit Link auf CDP einblenden (s. Kapitel 2.2.4.6)

```
void OnMouseDown()
     {
      //Wenn Text und Button für die angetippte Komponente aktiv (=sichtbar) sind
       if (abc.activeInHierarchy == true
            {
            …
            }
      //Wenn Text und Button für die Komponente nicht aktiv (=unsichtbar) sind
      else
            {
            …
            }
      }
```

Dies wurde beispielhaft für die Komponenten Wassertank, Siebhalter und Milchaufschäumdüse umgesetzt. Da die Komponenten aus mehreren 3D-Objekten bestanden, wurde für jede Komponente eine Gruppe in der Objekthierarchie erstellt.

Analog zu OnMouseDown() gibt es auch die Funktion OnMouseUp(). Diese wurde im vorliegenden Projekt jedoch nicht verwendet, da parallel zur Einfärbung einer Komponente auch Textboxen angezeigt werden sollten, die bei OnMouseUp() verschwinden würden, sobald der Nutzer den Touchscreen nicht mehr berührt. Das wurde als nicht nutzerfreundlich empfunden.

Damit ein Klick auf ein Objekt mit der OnMouseDown()-Funktion registriert werden kann, muss dem Objekt, falls noch nicht vorhanden, eine Collider-Komponente (z. B. Mesh Collider) hinzugefügt werden.

Die Farbe von 3D-Objekten ist über die Renderer-Komponente des Objekts änderbar. Da für Wassertank, Siebhalter und Milchaufschäumdüse jeweils eine Objektgruppe erstellt wurde, war eine foreach-Schleife nötig, die die Renderer der Kinder der Objektgruppe anspricht und deren Farbe in diesem Fall auf Rot setzt:

```
foreach(Renderer r in GetComponentsInChildren<Renderer>())
            {
            r.material.color= Color.red;
            }
```

OnMouseDown() kann für ein Objekt deaktiviert werden, indem es auf die Ebene „Ignore Raycast“ verschoben wird:

```
foreach (Transform trans in obj.GetComponentsInChildren<Transform>(true)) {
            trans.gameObject.layer = LayerMask.NameToLayer("Ignore Raycast");
            }
```

So konnte in der fertigen App sichergestellt werden, dass die Komponenten der Kaffeemaschine nur angetippt werden können, wenn die Schrittanleitung eingeblendet ist. Bei Anzeige der Schrittübersicht sollten die Komponenten nicht antippbar sein, damit das Textfeld mit der Kurzbeschreibung der Komponente nicht die Schrittübersicht überlagert. Durch Verschieben des Korpus der Kaffeemaschine auf die Ebene „Ignore Raycast“ konnte auch sichergestellt werden, dass der Wassertank sich nur bei Berührung des Wassertanks einfärbt, und nicht bei Berührung des Korpus, der sich vor dem Wassertank befindet.

Durch Verschieben der Objekte zurück auf die Ebene „Default“ werden sie wieder antippbar gemacht:

```
foreach (Transform trans in obj.GetComponentsInChildren<Transform>(true)) {
        trans.gameObject.layer = LayerMask.NameToLayer("Default");
        }
```

2.2.4.8 3D-Modelle ein- und ausblenden

3D-Modelle oder Teile davon können über ihre Renderer-Komponente (hier: Mesh Renderer) eingeblendet (enabled = true) und ausgeblendet (enabled = false) werden:

```
GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
```

Diese Funktion wurde verwendet, um Teile des 3D-Modells (z. B. Kaffeetasse) passend zu dem aktuell angezeigten Schritt der Anleitung in der AR-App aus- oder einblenden zu können.

2.2.4.9 Animationen erstellen und abspielen

Schrittanleitungen sollten in der AR-App neben den bereits genannten Manipulationen von 3D-Modellen oder UI-Objekten durch Animationen unterstützt werden.

Animationen können direkt in Unity mit der Animation-Komponente (s. Abb. 17) erstellt werden. Das in der Objekthierarchie ausgewählte Objekt kann zu einem aus der Zeitleiste auswählbaren Zeitpunkt beliebig verändert, d. h. Rotation, Position und Skalierung angepasst werden. Um die Animation zu prüfen, kann diese über die Play-Taste wie ein Film abgespielt werden.

Da im vorliegenden Projekt mehrere Teile eines 3D-Modells unabhängig voneinander animiert werden sollten, war es sinnvoll, alle Animationen für den Elternknoten „testmodell“ des Modells zu erstellen. Auf diese Weise können die Animationen im C#-Skript später einfacher angesprochen werden, nämlich immer über denselben Elternknoten. Wenn eine Animation für den Elternknoten „testmodell“ erstellt wird, kann man über eine Baumstruktur (s. Abb. 18) auswählen, ob die Animation (z. B. Rotation) sich auf das gesamte Objekt oder nur einen Teil davon beziehen soll.

Animationen erstellen

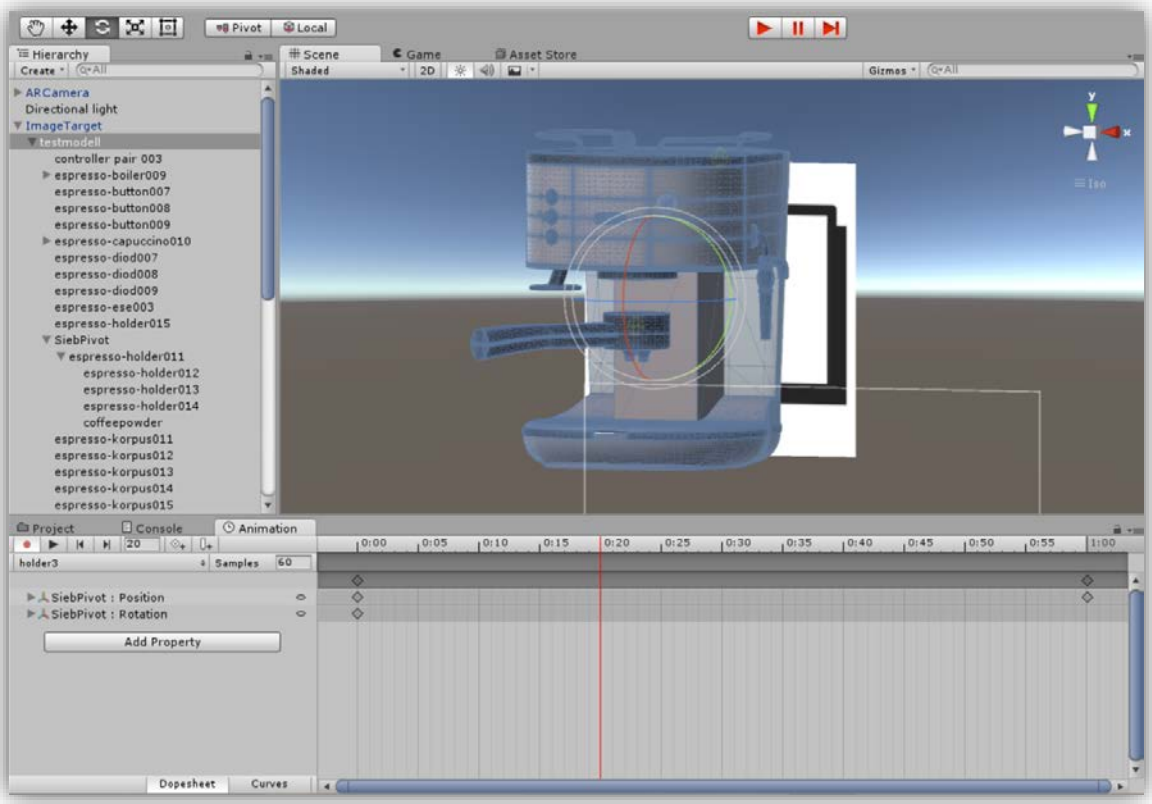

Abb. 17: Animationen erstellen mit der Animation-Komponente

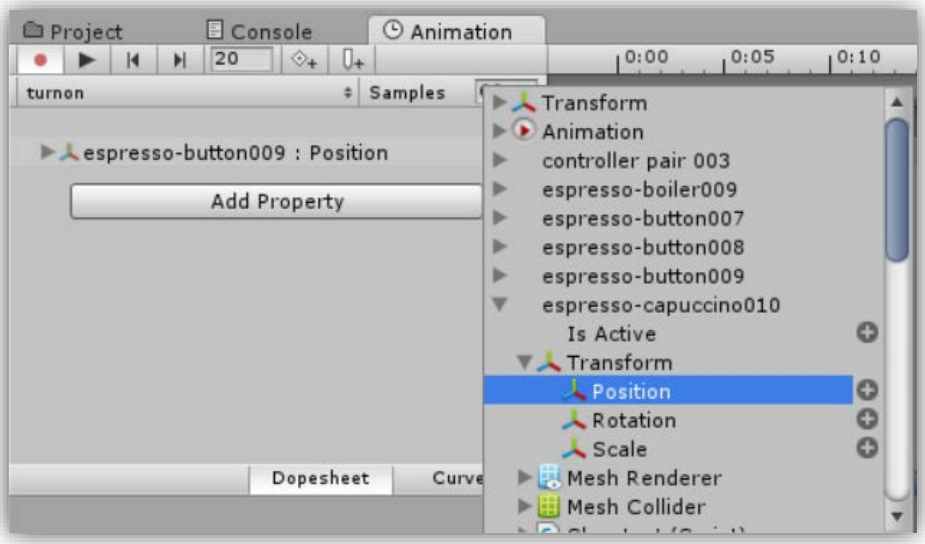

Abb. 18: Auswahl, welche Komponente von "testmodell" animiert werden soll (hier: espresso-capuccino010)

Beim Rotieren eines Objekts ist darauf zu achten, in der Menüzeile oben links die gewünschte Rotationsart zu wählen:

- Center: Das Objekt wird um sein eigenes Zentrum gedreht
- Pivot: Das Objekt wird um einen Drehpunkt gedreht

Wenn man die Option Pivot wählt, wurde der Drehpunkt, um den das Objekt rotiert, i. d. R. bereits bei Erstellung des 3D-Modells im 3D-Programm festgelegt und ist somit in Unity nicht direkt änderbar. Dafür gibt es ein Workaround, bei dem man für den gewünschten Drehpunkt ein leeres Objekt erstellt und den Drehpunkt auf diese Weise exakt in der Szene positioniert. Um den Drehpunkt besser erkennen und somit besser positionieren zu können, kann ein sogenanntes Gizmo-Skript verwendet werden, das den Drehpunkt, hier mit einer gelben Kugel, hervorhebt. Das Objekt, das animiert werden soll, wird anschließend dem leeren Objekt untergeordnet, sodass das leere Objekt als Drehpunkt dient. Das verwendete Gizmo-Skript stammt vom YouTuber unitycookie (YouTube 2013).

```
using UnityEngine;
using System.Collections;

public class Gizmo : MonoBehaviour {

	public float gizmoSize = .75f;
	public Color gizmoColor = Color.yellow;

	void OnDrawGizmos()
	{
		Gizmos.color = gizmoColor;
		Gizmos.DrawWireSphere(transform.position, gizmoSize);
	}
}
```

Abb. 19: Gizmo-Skript

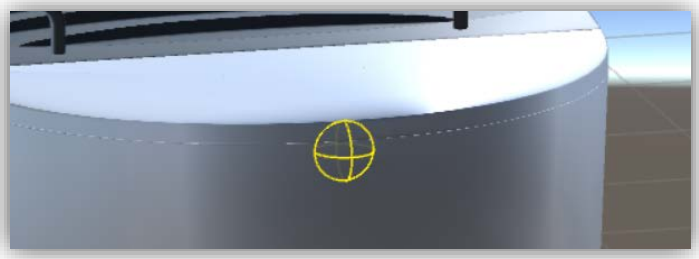

Abb. 20: Durch Gizmo-Skript erstellte Kugel zur Visualisierung des Drehpunktes

Die erstellten Animationen müssen anschließend (falls nicht bereits automatisch geschehen) dem entsprechenden Objekt zugeordnet werden. Dazu wird im Inspektor des Objekts eine neue Animation-Komponente angelegt und eine oder mehrere Animationen über Drag & Drop hineingezogen. Damit die Animationen über ein C#-Skript korrekt abgespielt werden können, muss in der Animation-Komponente die Option „Play Automatically“ ausgeschaltet sowie der Culling Type auf „Based on Renderers“ gesetzt werden. Außerdem müssen die Animationen in der Debug-Ansicht im Inspektor der Animation auf den Typ Legacy gestellt werden.

Alternativ zur Animation-Komponente kann die sehr ähnlich bezeichnete Animator-Komponente zur Verwaltung und Verknüpfung von Animationen verwendet werden. Für dieses Projekt war es jedoch ausreichend, die Animationen mithilfe der Animation-Komponente dem Objekt zuzuordnen und anschließend mit C# zu verwalten. In jedem Falle ist darauf zu achten, dass ein Objekt entweder eine Animation-Komponente oder eine Animator-Komponente enthält und nicht beides – ansonsten kommt es zu Fehlermeldungen.

Animationen abspielen

Für das Abspielen von Animationen gibt es in Unity sehr viele Funktionen – sowohl für die Animator-Komponente als auch für die Animation-Komponente. Auf die Unterscheidung zwischen den beiden Komponenten ist deshalb auch bei der Programmierung in C# zu achten.

Da es in der AR-App möglich sein sollte, sich die Animationen sowie den passenden Schritt der Anleitung einblenden zu lassen, wurde mit einem Weiter-Button und einem Zurück-Button gearbeitet, mit denen der Anwender von Schritt zu Schritt navigieren kann. Um mitverfolgen zu können, welcher Schritt gerade aktiv ist, wurde ein Zähler verwendet: Bei Klick auf den Weiter-Button wurde der Zähler um 1 erhöht; bei Klick auf den Zurück-Button um 1 verringert. Je nachdem, welcher Schritt gerade aktiv ist (was durch eine switch-Funktion abgefragt wird), soll eine andere Animation abgespielt sowie ein anderer Schritt als Textbox eingeblendet werden (zum Ein- und Ausblenden von Textboxen s. Kapitel 2.2.4.6).

Die einfachste Funktion, um Animationen abzuspielen, ist Play():

```
anim = gameobj.GetComponent<Animation>();
anim.Play("holder4");
```

Wenn mehrere Animationen mit Play() abgespielt werden, werden diese gleichzeitig abgespielt. Um eine Sequenz von Animationen abzuspielen, ist die Funktion PlayQueued() notwendig. Mit dieser Funktion wird eine Animation erst abgespielt, wenn alle anderen Animationen bereits bis zu ihrem Ende abgespielt worden sind:

```
anim = gameobj.GetComponent<Animation>();
anim.PlayQueued("turnon", QueueMode.CompleteOthers);
```

Alternativ ist auch die Funktion CrossFadeQueued() möglich, die aufeinanderfolgende Animationen ineinander überblendet:

```
anim = gameobj.GetComponent<Animation>();
anim.CrossFadeQueued("holder3", 0.3F, QueueMode.PlayNow);
```

In der AR-App sollte es außerdem möglich sein, zum vorherigen Schritt zurück zu springen, falls der Anwender sich etwas noch einmal ansehen möchte. Da die Animationen auf der Endposition des 3D-Modells der vorherigen Animation aufbauen, war es notwendig, beim Zurückspringen die soeben abgespielte Animation rückgängig zu machen. Wenn beispielsweise zu Schritt 2 zurückgesprungen wird, muss die Animation von Schritt 3 rückgängig gemacht werden, damit die Animation von Schritt 2 bei der richtigen Ausgangsposition des 3D-Modells startet.

Um eine Animation zurückzusetzen, d. h. rückwärts abzuspielen, muss lediglich die Geschwindigkeit der Animation negativ gesetzt werden. Beim Wert -1 wird die Animation in normaler Schnelligkeit rückwärts abgespielt. Da die Animation hier aber sofort zurück auf die Ausgangsposition gesetzt werden sollte, wurde der Wert -20 gewählt, d. h. die Animation wird mit 20-facher Geschwindigkeit rückwärts abgespielt. Damit die Animation von hinten nach vorne abgespielt wird, muss außerdem die Startzeit der Animation der Endzeit (entspricht der Gesamtlänge) der Animation gleichgesetzt werden, und schließlich die Animation mit Play() abgespielt werden:

```
anim = gameobj.GetComponent<Animation>();
anim["boiler4"].speed = -20;
anim["boiler4"].time = anim["boiler4"].length;
anim.Play("boiler4");
```

2.2.4.10 App erstellen und auf das Endgerät laden

Um den Build der App zu ermöglichen, muss in Unity in den Voreinstellungen sowohl der Speicherort der Android SDK als auch der JDK angegeben werden.

Damit Apps auf über USB angeschlossene Geräte geladen werden können, muss in den Entwickleroptionen des Geräts das USB-Debugging aktiviert werden. Außerdem sollte vor dem Build der App überprüft werden, ob das Gerät richtig erkannt wird (auch dazu ist der Debugging-Modus nötig). Dies kann z. B. über das Android Platform Tool adb.exe, welches sich im Android-SDK-Verzeichnis AndroidSDK\platform-tools befindet, überprüft werden. Dazu muss über die Kommandozeile in das SDK-Verzeichnis navigiert und der Befehl „adb.exe devices“ eingegeben werden. Bei erfolgreicher Autorisierung erscheint das Gerät in der „List of devices attached“.

In den Player Settings für Android können neben den in Kapitel 2.2.4.4 bereits angesprochenen Einstellungen für die Display-Orientierung weitere Einstellungen wie Icon, Splash Image oder Einstellungen für das Rendering geändert werden. Wichtig ist insbesondere, einen Bundle Identifier anzugeben, der sich aus Produktname und Firmenname zusammensetzt, die ebenfalls angegeben werden müssen. Außerdem muss im Inspektor der ARCamera ein App License Key eingegeben werden, der im Vuforia Developer Portal erhältlich ist.

Wenn alle diese Voraussetzungen erfüllt sind, lässt sich die App einfach über die Funktion File > Build & Run erstellen und auf das Gerät laden.

2.2.5 Exkurs: HTML-Inhalte in der AR-App anzeigen

Abschließen wurde untersucht, wie die derzeit auf zwei Apps (Browser für CDP und AR-App) aufgeteilte Anwendung zu einer einzigen App zusammengeführt werden könnte. So müsste Anwender nicht zwischen Browser und AR-App hin- und herspringen, sondern könnte die HTML-Inhalte direkt neben dem 3D-Modell in der AR-App anzeigen. Dafür müssten die HTML-Inhalte direkt in der App angezeigt werden können.

Dies wird in Unity an sich nicht unterstützt, ist prinzipiell ber mit Unity-Plugins von Drittanbietern möglich, wie Awesomium, HTMLTexture, uWebKit oder Coherent UI. uWebKit wird jedoch leider mittlerweile nicht mehr entwickelt, HTMLTexture läuft derzeit nur unter Mac, Coherent UI und einige andere Plugins sind nur in Unity Pro anwendbar und kamen somit für das vorliegende Projekt nicht in Frage. Awesomium wurde für das Projekt installiert und getestet. Jedoch treten beim Testen der Anwendungen Fehlermeldungen auf, die auch vielen anderen Usern Probleme bereiten und für die es auch nach einiger Recherche derzeit keine Lösung zu geben scheint. Die Informationen auf http://labs.awesomium.com/ sind ebenfalls veraltet, wie z. B. das Tutorial zur Einbindung von Awesomium in Unity, das sich auf ältere Versionen von Awesomium und auch Untiy bezieht.

3 Bewertung von Unity/Vuforia

An dieser Stelle sollen die in Kapitel 2.2.4 beschriebenen Probleme noch einmal zu einer Gesamtbewertung zusammengefasst werden.

Obwohl zu Beginn des Projekts keinerlei praktische Vorkenntnisse zu Augmented Reality vorhanden waren, konnten alle Ziele für die AR-App wie gewollt umgesetzt werden. Durch die vorhandenen Kenntnisse in JavaScript und die Fülle an Informationen im Internet zu Unity und C# allgemein war die Einarbeitung in C# kein größeres Hindernis.

Dass Unity so ein mächtiges, professionelles Tool zur Entwicklung von 3D-Anwendungen mit großem Funktionsumfang ist, macht es einem Anfänger jedoch schwer, sich auf eigene Faust in die Materie einzuarbeiten. Allein in der offiziellen Unity-Hilfe gibt es sehr viele Informationen, die durchforstet werden müssen, bis man die Funktion gefunden hat, nach der man gesucht hat. Dies wird dadurch erschwert, dass es viele ähnliche Funktionen gibt, etwas zu tun, die zusätzlich in vielen weiteren Informationsquellen verteilt sind. Dies hat teilweise sehr viel Zeit in Anspruch genommen.

Etwas Eingewöhnung erfordert auch die Zuweisung der C#-Skripte an bestimmte Objekte. Welches C#-Skript an welches Objekt gehängt werden muss, um das gewünschte Verhalten zu erreichen, war zu Beginn verwirrend. Auch, dass man den Variablen nicht direkt in C# Werte zuweisen muss, sondern dies später über Drag & Drop im Inspektor in Unity machen kann, war ungewohnt, bringt aber einige Vorteile wie z. B. die Wiederverwendbarkeit von Skripten, denen je nach Bezugsobjekt ein anderer Wert für Variablen zugewiesen werden kann.

Die 3D-Objekterkennung ist ebenfalls problematisch. Für ein erfolgreiches Scannen sind sehr gute Lichtverhältnisse notwendig, da sonst nicht genug Anhaltspunkte erkannt werden. Zudem ist das Scannen von Objekten nur bis zu einer bestimmten Größe möglich. Die im Projekt verwendete Kaffeemaschine ist vermutlich schon zu groß, da das Scannen mit dem Object Scanner nicht funktioniert hat und nur sehr wenige Anhaltspunkte erkannt wurden.

Ebenfalls nicht optimal sind die Möglichkeiten zur Formatierung von Texten direkt in Unity. Schriftart, Schriftstil und Schriftgröße können nur für das ganze Text-Objekt und nicht für einzelne Zeichen eingestellt werden. Problematisch ist außerdem, dass die Schrift mitskaliert und damit verzerrt wird, wenn beispielsweise ein Button nach Hinzufügen der Schrift größer skaliert wird.

Das responsive Verhalten der UI-Elemente wird zudem nicht, wie aus der Web-Programmierung gewohnt, über prozentuale Größenangaben gesteuert, sondern über spezielle Anker, die manuell und nach Augenmaß direkt in der Szene gesetzt werden, was unnötig kompliziert und unhandlich erscheint.

In der fertigen App werden die Animationen mit kleinen Ungenauigkeiten abgespielt. Beispielsweise klappt der Deckel des Wassertanks nicht vollständig zu, obwohl die entsprechende Animation in Unity richtig abspielt und der Deckel vollständig schließt. Das größte Hindernis bei der Arbeit mit Unity war jedoch, dass bei jeder Änderung eines beliebigen C#-Skriptes plötzlich mehrfach die Fehlermeldung „error CS0234: The type or namespace name `UI' does not exist in the namespace `UnityEngine'. Are you missing an assembly reference?“ angezeigt wurde – selbst bei von Unity oder Vuforia mitgelieferten Skripten, an denen nie etwas geändert wurde. Zudem wurden alle C#-Skripte direkt in Unity erstellt und nicht importiert, und die Fehlermeldungen verschwanden nach dem Beenden und erneuten Öffnen von Unity. Da die AR-App darüber hinaus einwandfrei funktionierte, kann vermutet werden, dass die Ursache für die Fehler nicht in der erstellen AR-App selbst liegt.

4 Fazit und Ausblick

Alle zu Beginn des Projekts definierten Ziele konnten erreicht werden. Zunächst wurde ein funktionierendes Content-Delivery-Portal mit allen gewünschten Funktionen wie der facettierten Suche, Volltextsuche und ein- und ausblendbaren Viewlets erstellt. Abgesehen von dem eingesetzten Framework EasyUI für die ein- und ausblendbaren Viewlets sowie Bootstrap für die Formatierung wurde das jQuery vollständig selbst programmiert.

Auch bei der Erstellung der AR-App konnten alle geplanten Funktionen erfolgreich umgesetzt werden. Darüber hinaus sind noch weitere Funktionen denkbar, die aus Zeitgründen in diesem Projekt nicht umgesetzt werden konnten. Wenn beide Apps zu einer vereint werden, könnte man z. B. auch verschiedene Schrittanleitungen und zugehörige Animationen für verschiedene Zielgruppen unterscheiden. Ideal wäre, wenn der Anwender in der App einstellen könnte, zu welcher Nutzergruppe (z. B. Bediener, Wartungsmitarbeiter) er gehört. Als spezieller Content für die Zielgruppe Wartungsmitarbeiter wäre z. B. eine Animation denkbar, die alle Bauteile wie in einer Explosionszeichnung anordnet.

Außerdem könnte versucht werden, Objekterkennung (statt der aktuell umgesetzten Bilderkennung) einzusetzen, ggf. mit einem anderen Tool, falls dies mit Vuforia aufgrund der beschriebenen Schwierigkeiten nicht möglich ist. Dann könnte die Kaffeemaschine idealerweise aus mehreren Blickwinkeln erkannt werden und das Problem, dass der Marker bei der Bilderkennung „verloren“ werden kann, wäre gelöst.

Zur Verwendung von AR in der Technischen Dokumentation kann gesagt werden, dass mithilfe von AR die Inhalte anschaulicher, interaktiver und damit verständlicher dargestellt werden können. Anstelle von Bilder und entsprechenden Legenden können Animationen verwendet werden, die innerhalb von Sekunden aus 3D-Modellen erstellt und ebenso schnell geändert werden können. Die Animationen können mit AR direkt auf das Produkt (z. B. eine Maschine. die gewartet werden muss) projiziert werden, sodass der Anwender Elemente wie einzubauende Komponenten und ihre korrekte Lage in der Maschine sofort erkennen kann. Die Bild- oder Objekterkennung hat außerdem den Vorteil, dass die App „weiß“, zu welchem Bauteil der Anwender gerade Informationen benötigt. So muss der Anwender nicht lange in einem Handbuch nach Informationen zu dem Bauteil suchen, sondern bekommt sofort die relevanten Informationen zum Bauteil angezeigt. Durch die Verwendung von Animationen ist auch weniger Text nötig, um Vorgänge zu beschreiben. Die Umsetzung einer gesamten, professionellen AR-App ist jedoch zunächst mit einem hohen initialen Aufwand verbunden, und je nach verwendetem AR-Tool kann es Einschränkungen in den Funktionalitäten geben.

5 Anhang: Kommentierter Code

Anmerkung: Hier werden ausschließlich Skripte aufgeführt, die für das Content-Delivery-Portal oder die App verwendet wurden. Skripte, die im Unity-Projekt zum Ausprobieren angelegt, aber nicht im fertigen Projekt, verwendet wurden, sind:

- color.cs
- colorgroup.cs
- play_anim_new.cs
- play_anim_on_ui_button.cs
- play_several_anim.cs
- update_anim.cs

portal.htm (Auszug)
Aufbau des Content-Delivery-Portals

```
<body>
   <div class="easyui-layout" style="width:100%; height:100%">
    //Suche-Viewlet
      <div region="west" split="true" title="Filter" style="width:20%;">
         <form>
            <div class="container" style="width:100%;">
               <div id="filters">
                  <div class="input-control">
                     <h5>Arbeitsphase:</h5>
                     <div class="checkbox">
                           <input id="check1" type="checkbox" name="check" value="Gesamt"
                           class="category1" style="margin-left: 0px;">
                           <label for="check1">Gesamt</label>
                     </div>
                     <div class="checkbox">
                           <input id="check2" type="checkbox" name="check" value="Inbetriebnahme"
                           class="category1" style="margin-left: 0px;">
                           <label for="check2">Inbetriebnahme</label>
                      </div>
                     […]
                     </div>
               </div>
            </div>
         </form>
      </div>
    //Topic-Anzeige-Viewlet
      <div id="content" class="target" region="center" title="Topic-Inhalt">
            <object style="width:100%; height:99%;" data="images/Delonghi_Icona.jpg"/>
       </div>
    //IHVZ-Viewlet
      <div region="east" split="true" title="Inhaltsverzeichnis" style="width:20%;">
         <ol class="rectangle-list">
            <li class="chapter" style="padding:3px;">
                  <a href="#" class="strong">Einleitung</a>
                  <ol style="margin-left:10px;">
                        <li>
                        <a href="#" id="Über_dieses_Dokument" class="ihvz">1.1 Über dieses
                              Dokument</a>
                        </li>
                        <li><a href="#" id="Bestimmungsgemäßer_Gebrauch" class="ihvz">1.2
                              Bestimmungsgemäßer Gebrauch</a>
                        </li>
                  </ol>
            </li>
            […]
         </ol>
      </div>
    //Suchergebnis-Viewlet
      <div id="south" region="south" split="true" title="Suchergebnisse" style="height:30%;">
         <div class="searchresults">
            <table class="table">
               <thead>…</thead>
               <tbody>
                  <tr class="resultblock" data-tag-info1="Gesamt" data-tag-info2="Beschreibung"
                   data-tag-prod="Tassenwärmer">
                     <td id="topictitle">
                           <a href="#" id="Tassenwärmer" class="ihvz">Tassenwärmer</a>
                     </td>
                     <td id="topicpclass">Tassenwärmer</td>
                     <td id="topiciclass1">Gesamt</td>
                     <td id="topiciclass2">Beschreibung</td>
                  </tr>
                   […]
               </tbody>
            </table>
         </div>
      </div>
   </div>
</body>
```

test.js
Programmierung des Content-Delivery-Portals

```
$(document).ready(function(){

        // Extrahiere den Zusatz in der URL (z.B. check18)
        var query = window.location.search.substring(1);

         // Erweiterung zu z.B. #check18 (ID des Werts der Facette)
         var checkid = "#" + query
         // Setze einen Haken beim Wert der Facette und Führe myFunction() aus
         $(checkid).prop('checked', true);
         myFunction();

         //Setze Schrift im IHVZ auf Schwarz
        $('a.ihvz').css("color", "rgb(0, 0, 0)");
         //Verstecke die zweite Ebene des IHVZ
        $('a.strong').next().hide();

        //Zähle die Suchergebnisse in der Tabelle und schreibe Sie in den Titel des Panels
        var cnt = $('tr:not([style*="display: none"])').length - 1;
      $( '.panel-title:last' ).html("Suchergebnisse: " + cnt);

      //Blende die 2. Ebene des IHVZ aus und ein bei Klick auf das zugehörige Kapitel erster Ebene
      $('a.strong').on('click',function(){
      if($(this).next().is('ol:hidden')){
                $(this).next('ol').show();
         }
         else{
                $(this).next('ol').hide();
         }
    });

       $('a.ihvz').on('click',function(){
      var ID = $(this).attr('id');

             $('a.ihvz').css("color", "rgb(0, 0, 0)");

             var Inhalt = $(this).text();

             //Bei Klick auf einen Link zu einem Topic in der Suchergebnisliste, färbe den Link
             sowohl im IHVZ als auch in der Suchergebnisliste rot
             $('a.ihvz:contains("'+Inhalt+'")').css("color", "rgb(220, 20, 60)");
             $(this).css("color", "rgb(220, 20, 60)");

             //Springe außerdem zur richtigen Stelle im IHVZ und klappe alle anderen Kapitel zu
             $('a.ihvz:contains("'+Inhalt+'")').parent('li').parent('ol').parent('li').siblings()
                    .children('ol').hide();
             $('a.ihvz:contains("'+Inhalt+'")').parents('ol').show();

             //Zeige das Topic im Object-Element an
             $("object").attr('data', ID + ".htm");
             $("object").attr('style', "width:100%;height:99%");
      });

      //Schreibe hinter Facettenwert die Anzahl der dafür vorhandenen Ergebnisse (.resultblock)
      $('.category1, .category2, .category3').each(function(){
       var category = $(this).val();
             var categoryid = $(this).attr("id");
             var len1 = $('.resultblock').filter(function() {
                    return $(this).attr('data-tag-info1') == category;
                    }).length;
             var len2 = $('.resultblock').filter(function() {
                    return $(this).attr('data-tag-info2') == category;
                    }).length;
             var len3 = $('.resultblock').filter(function() {
                    return $(this).attr('data-tag-prod') == category;
                    }).length;
             var len = len1 + len2 + len3
             $('label[for="'+categoryid+'"]').html($(this).val() + ' (' + len + ')' );
     });

      //Wenn die Facettenwerte geändert werden, führe myFunction() aus
      $('input').change(function(){
             myFunction();
      });
```

```
        function myFunction(){

        //Erstelle einen Array für jede Kategorie
         var category_list1 = [];
         var category_list2 = [];
         var category_list3 = [];

       //Schiebe angewählte Werte in den jeweiligen Array
      $('.category1:checked').each(function(){
        var category1 = $(this).val();
              category_list1.push(category1);
      });

         $('.category2:checked').each(function(){
        var category2 = $(this).val();
        category_list2.push(category2);
      });

         $('.category3:checked').each(function(){
        var category3 = $(this).val();
        category_list3.push(category3);
      });

       //Blende Zeilen aus der Suchergebnisliste (Klasse resultblock) aus oder ein, je nachdem, ob
       ein passender Facettenwert ausgewählt wurde (Matching über data-tag-Attributen)
       //Blende Facettenwerte aus, wenn es kein passendes Ergebnis in der Suchergebnisliste dazu
       gibt. Ansonsten blende die Facette ein.

        $('.resultblock').each(function(){

              var item1 = $(this).attr('data-tag-info1');
              var item2 = $(this).attr('data-tag-prod');
              var item3 = $(this).attr('data-tag-info2');

       if(category_list1.length == 0 && category_list3.length == 0 && category_list2.length != 0 &&
              jQuery.inArray(item2,category_list2) > -1){
              $(this).fadeIn('slow');
              $('.category1[value="'+item1+'"], .category2[value="'+item2+'"],
                     .category3[value="'+item3+'"]').fadeIn('slow');
              $('.category1[value="'+item1+'"], .category2[value="'+item2+'"],
                     .category3[value="'+item3+'"]').next().fadeIn('slow');
                }
                else if(category_list1.length == 0 && category_list2.length == 0 &&
                     category_list3.length != 0 && jQuery.inArray(item3,category_list3) > -1){
                     […]
                }
              […]
          else{
              $(this).hide();
              }
        });

       $('.category1, .category2, .category3').each(function(){
       //Wenn kein Facettenwert angewählt ist, schreibe hinter jeden Facettenwert (Klasse
       category1,2,3) die Anzahl der insgesamt dafür vorhandenen Ergebnisse (Klasse resultblock)
       if(category_list1.length == 0 && category_list3.length == 0 && category_list2.length == 0){
                     var category = $(this).val();
                     var categoryid = $(this).attr("id");

                     $('.checkbox input[id="'+categoryid+'"]').show();
                     $('.checkbox label[for="'+categoryid+'"]').show();

                     var len1 = $('.resultblock').filter(function() {
                            return $(this).attr('data-tag-info1') == category;
                            }).length;
                     var len2 = $('.resultblock').filter(function() {
                            return $(this).attr('data-tag-info2') == category;
                            }).length;
                     var len3 = $('.resultblock').filter(function() {
                            return $(this).attr('data-tag-prod') == category;
                            }).length;
                     var len = len1 + len2 + len3
                     $('label[for="'+categoryid+'"]').html($(this).val() + ' (' + len + ')' );
                }
       //Wenn ein oder mehrere Facettenwerte angewählt sind, überprüfe die Anzahl der dafür noch
       verbleibenden Ergebnisse (tr:visible)
       else{
```

```
                    var category = $(this).val();
                    var categoryid = $(this).attr("id");
                    var len1 = $('tr:visible').filter(function() {
                           return $(this).attr('data-tag-info1') == category;
                           }).length;
                    var len2 = $('tr:visible').filter(function() {
                           return $(this).attr('data-tag-info2') == category;
                           }).length;
                    var len3 = $('tr:visible').filter(function() {
                           return $(this).attr('data-tag-prod') == category;
                           }).length;
                    var len = len1 + len2 + len3

      //Wenn passende Ergebnisse vorhanden sind, schreibe diese in Klammern hinter den Facettenwert
                    if (len1 + len2 + len3 != 0){
                    $('label[for="'+categoryid+'"]').html($(this).val() + ' (' + len + ')' );
                    }
      //Wenn keine passenden Ergebnisse vorhanden sind, verstecke den Facettenwert
                    else{
                           $('.checkbox label[for="'+categoryid+'"]').hide();
                           $('.checkbox input[id="'+categoryid+'"]').hide();
                    }

              }
     });

      //Zähle die Suchergebnisse in der Tabelle und schreibe Sie in den Titel des Panels
      var cnt = $('tr:not([style*="display: none"])').length - 1;
      $( '.panel-title:last' ).html("Suchergebnisse: " + cnt);

   }



$('.searchbutton').on('click',function(){

       // Wert des Eingabefelds in Variable speichern, Ergebnis-Zahl auf 0 setzen
      var id = $(this).attr('id');
      var volltextsuche = $("#volltextsuche").val() , count = 0;

      // Wenn auf den Zurück-Button geklickt wurde, leere das Suchfeld und führe myFunction aus
      if(id == 'zurueck'){
             $("#volltextsuche").val('');
             myFunction();
      }
      //Wenn auf den Weiter-Button geklickt wurde
      else{
       // Durchsuche vorhandene Suchergebnisse (=sichtbare Tabellenzeilen)
       $("tr:visible td:first-child").each(function(){
           // Blende unpassende Zeilen aus
           if ($(this).text().search(new RegExp(volltextsuche, "i")) < 0) {
               $(this).parent().fadeOut();
           // Blende passende Zeilen ein und erhöhe die Ergebnis-Zahl
           } else {
               $(this).parent().show();
               count++;
           }
       });

       // Schreibe die Ergebnis-Zahl in den Panel-Titel
       var numberItems = count;
       $('.panel-title:last').html("Suchergebnisse: "+count);

             }
   });

 });
```

closeanl.cs

Bei Klick des Schließen-Buttons in der Schrittanleitung: Schließen der Schrittanleitung, Einblenden der Schrittübersicht. Die 3 Komponenten, zu denen es eine Info-Box gibt, auf eine andere Ebene setzen, damit sich die Infobox beim Berühren der Komponenten nicht mit der Schrittanleitung überlagert.

```
public class closeanl : MonoBehaviour {

	//private Animation anim;
	//private AnimationState newstate;
	//private AnimationState newstate2;
	//private AnimationState newstate3;
	//private AnimationState newstate4;
	//public GameObject gameobj;
	public GameObject obj;
	public GameObject obj2;
	public GameObject obj3;

	public void resetanl ()
	{
		//Schrittanleitung ausblenden, Schrittübersicht einblenden
		GameObject.Find("Panel").GetComponent<CanvasGroup>().alpha = 0;
		GameObject.Find("Panel").GetComponent<CanvasGroup>().interactable = false;
		GameObject.Find("Panel2").GetComponent<CanvasGroup>().alpha = 1;
		GameObject.Find("Panel2").GetComponent<CanvasGroup>().interactable = true;

		//anim = gameobj.GetComponent<Animation>();

		//3D-Modell in die Ausgangsposition zurückbringen

		/* newstate = anim.PlayQueued("boiler3"); //, QueueMode.PlayNow)
		newstate.speed = 15;
		newstate.time = 0.0f;

		newstate2 = anim.PlayQueued("boiler4"); //, QueueMode.PlayNow)
		newstate2.speed = 15;
		newstate2.time = 0.0f;

		newstate3 = anim.PlayQueued("holder3"); //, QueueMode.PlayNow)
		newstate3.speed = 15;
		newstate3.time = 0.0f;

		newstate4 = anim.PlayQueued("holder4"); //, QueueMode.PlayNow)
		newstate4.speed = 15;
		newstate4.time = 0.0f; */

		//LEDs "ausschalten", d.h. schwarz färben
		/* GameObject.Find("espresso-diod008").GetComponent<Renderer>()
			.material.SetColor("_Color", Color.black);
		GameObject.Find("espresso-diod009").
			GetComponent<Renderer>().material.SetColor("_Color", Color.black); */

		//Teile, die nicht zur Kaffeemaschine gehören, ausblenden
		/* GameObject.Find("coffeepowder").GetComponent<MeshRenderer>().enabled = false;
		GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = false;
		GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = false;
		GameObject.Find("output1").GetComponent<MeshRenderer>().enabled = false;
		GameObject.Find("output2").GetComponent<MeshRenderer>().enabled = false; */

		//Komponenten mit Info-Box auf andere Ebene setzen, damit nicht klickbar
		foreach (Transform trans in obj.GetComponentsInChildren<Transform>(true)) {
			trans.gameObject.layer = LayerMask.NameToLayer("Ignore Raycast");
		}

		foreach (Transform trans in obj2.GetComponentsInChildren<Transform>(true)) {
			trans.gameObject.layer = LayerMask.NameToLayer("Ignore Raycast");
		}

		foreach (Transform trans in obj3.GetComponentsInChildren<Transform>(true)) {
			trans.gameObject.layer = LayerMask.NameToLayer("Ignore Raycast");
		}
    }

}
```

DefaultTrackableEventHandler.cs (Auszug)

Vorgefertigtes Skript zum Verhalten der Anwendung in Bezug auf den Tracker. Wurde angepasst, um die Übersicht über die wichtigsten Schritte (nur beim ersten Erkennen des Markers) einzublenden und bei jedem starten des Skripts alle Teile des 3D-Modells ausgeblendet werden, die nur bei bestimmten Schritten eingeblendet werden sollen.

```
private void OnTrackingFound()
        {

       […]

            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " found");

                      //Objekte, die nicht zur Kaffeemaschine gehören, ausblenden
                      GameObject.Find("coffeepowder").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("output1").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("output2").GetComponent<MeshRenderer>().enabled = false;

                      //Aktionen, die nur beim ersten Erkennen des Trackers ausgeführt werden sollen
                      if(firststarted==0)
                      {
                      Debug.Log("Allererster Start!");
                      //Schrittanleitung ausblenden
                      GameObject.Find("Panel").GetComponent<CanvasGroup>().alpha = 0;
                      GameObject.Find("Panel").GetComponent<CanvasGroup>().interactable = false;
                      //Schrittübersicht einblenden
                      GameObject.Find("Panel2").GetComponent<CanvasGroup>().alpha = 1;
                      GameObject.Find("Panel2").GetComponent<CanvasGroup>().interactable = true;

                      //Variable firststarted auf 1 setzen, damit oben stehender Code nur beim
                      ersten Erkennen ausgeführt wird
                      firststarted=1;
                      }

                      //Variable für aktuellen Schritt aus Schrittanl-Skript holen
                      mycounter2 = GameObject.Find("MeineUI").GetComponent<schrittanl>().mycounter;

                      //Je nach aktuellem Schritt andere Objekte einblenden, die nicht zur
                      Kaffeemaschine gehören
                      switch (mycounter2)
                             {
                      case 15:
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                      GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      case 14:
                      GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
                      GameObject.Find("output1").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("output2").GetComponent<MeshRenderer>().enabled = false;
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      case 13:
                      GameObject.Find("output1").GetComponent<MeshRenderer>().enabled = true;
                      GameObject.Find("output2").GetComponent<MeshRenderer>().enabled = true;
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      case 12:
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      case 11:
                      GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      case 8:
                      GameObject.Find("coffeepowder").GetComponent<MeshRenderer>().enabled = true;
                      break;
                      default:
                             Debug.Log("Ist nicht zwischen 1 und 15");
                             break;
                      }

        }
```

Gizmo.cs

Hervorhebung des ansonsten unsichtbaren Drehpunktes für Animationen mit der Einstellung „Pivot“

```
using UnityEngine;
using System.Collections;

public class Gizmo : MonoBehaviour {

	public float gizmoSize = .75f;
	public Color gizmoColor = Color.yellow;

	void OnDrawGizmos()
	{
		Gizmos.color = gizmoColor;
		Gizmos.DrawWireSphere(transform.position, gizmoSize);
	}
}
```

schrittanl.cs (Auszug)

Ein- und Ausblenden von UI-Elementen, Animationen und Teilen des 3D-Modells passend zum aktuellen Schritt der Schrittanleitung.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class schrittanl : MonoBehaviour {

	public int mycounter=0;
	public GameObject gameobj;
	private Animation anim;
	private GameObject steps = null;
	public Text activestep;
	/* public UnityEngine.Renderer rend1; */
	public Renderer rend2;

	//Funktion bei Klick auf Weiter-Button
	public void increase_counter ()
	{
		//Hochzählen der Variable bei Klick auf Weiter-Button, um aktuellen Schritt
		mitverfolgen zu können
		if(mycounter < 15)
		{
		mycounter++;
		}
		else{
		mycounter=1;
		}

		Debug.Log("Counter ist jetzt " + mycounter);

		//Werte für Variablen zuweisen
		anim = gameobj.GetComponent<Animation>();
		steps = GameObject.Find("Schritte");

		//Alle Schrittanleitungstexte ausblenden
		foreach(Text r in steps.GetComponentsInChildren<Text>()){
			r.enabled = false;
			}

		//Je nach aktuellem Schritt andere Objekte, Texte und Animationen einblenden
		switch (mycounter)
		{
		case 15:
			Debug.Log("Schritt 15");
			anim.PlayQueued("turnon", QueueMode.CompleteOthers);
			activestep = GameObject.Find("Schritt15").GetComponent<Text>();
			activestep.enabled = true;
			GameObject.Find("espresso-diod009").GetComponent<Renderer>().material.
				SetColor("_Color", Color.black);
			GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
			GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
			break;
		case 14:
			Debug.Log("Schritt 14");
			anim.PlayQueued("coffeeon", QueueMode.CompleteOthers);
			activestep = GameObject.Find("Schritt14").GetComponent<Text>();
			activestep.enabled = true;
			GameObject.Find("espresso-diod008").GetComponent<Renderer>().material.
				SetColor("_Color", Color.black);
			GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
			GameObject.Find("output1").GetComponent<MeshRenderer>().enabled = false;
			GameObject.Find("output2").GetComponent<MeshRenderer>().enabled = false;
			GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
		break;

		[…]

		default:
			Debug.Log("Ist nicht zwischen 1 und 15");
			break;
		}
	}
```

```
        //Funktion bei Klick auf Zurück-Button
        public void decrease_counter ()
        {
                //Vermindern der Variable bei Klick auf Zurück-Button, um aktuellen Schritt
                mitverfolgen zu können
                if(mycounter > 1)
                {
                mycounter--;
                }
                else{
                mycounter=15;
                }

                Debug.Log("Counter ist jetzt " + mycounter);

                //Werte für Variablen zuweisen
                anim = gameobj.GetComponent<Animation>();
                steps = GameObject.Find("Schritte");

                //Alle Schrittanleitungstexte ausblenden
                foreach(Text r in steps.GetComponentsInChildren<Text>()){
                        r.enabled = false;
                        }

                //Je nach aktuellem Schritt andere Objekte, Texte und Animationen einblenden
                switch (mycounter)
                {
                case 15:
                        Debug.Log("Schritt 15");
                        /* Animation des zu wiederholenden Schrittes abspielen*/
                        anim.CrossFadeQueued("turnon", 0.3F, QueueMode.PlayNow);

                        /* Passenden Schritt der Anleitung anzeigen */
                        activestep = GameObject.Find("Schritt15").GetComponent<Text>();
                        activestep.enabled = true;
                        GameObject.Find("espresso-diod009").GetComponent<Renderer>().material.
                                SetColor("_Color", Color.black);
                        GameObject.Find("cup").GetComponent<MeshRenderer>().enabled = true;
                        GameObject.Find("coffee").GetComponent<MeshRenderer>().enabled = true;
                        break;

                        […]

                case 9:
                        Debug.Log("Schritt 9");
                        /* Animation des nächsten Schrittes rückgängig machen */
                        anim["holder4"].speed = -20;
                        anim["holder4"].time = anim["holder4"].length;
                        anim.Play("holder4");

                        /* Animation des zu wiederholenden Schrittes abspielen*/
                        anim.CrossFadeQueued("holder3", 0.3F, QueueMode.PlayNow);

                        /* Passenden Schritt der Anleitung anzeigen */
                        activestep = GameObject.Find("Schritt9").GetComponent<Text>();
                        activestep.enabled = true;
                        break;
                case 8:
                        Debug.Log("Schritt 8");
                        /* Animation des nächsten Schrittes rückgängig machen */
                        anim["holder3"].speed = -20;
                        anim["holder3"].time = anim["holder3"].length;
                        anim.Play("holder3");

                        /* Animation des zu wiederholenden Schrittes abspielen
                        anim.CrossFadeQueued("boiler2", 0.3F, QueueMode.PlayNow);*/

                        /* Passenden Schritt der Anleitung anzeigen */
                        activestep = GameObject.Find("Schritt8").GetComponent<Text>();
                        activestep.enabled = true;
                        GameObject.Find("coffeepowder").GetComponent<MeshRenderer>().enabled = true;
                        break;
                default:
                        Debug.Log("Ist nicht zwischen 1 und 15");
                        break;
                }
        }
}
```

showpanel.cs

Einblenden der Schrittanleitung und Ausblenden der Übersicht mit den wichtigsten Schritten

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class showpanel : MonoBehaviour {

        // private Animation anim;
        public Text activestep;
        private GameObject steps = null;
        public GameObject obj;
        public GameObject obj2;
        public GameObject obj3;

        public void switchpanels ()
        {
                //Schrittanleitung einblenden
                 GameObject.Find("Panel").GetComponent<CanvasGroup>().alpha = 1;
                 GameObject.Find("Panel").GetComponent<CanvasGroup>().interactable = true;
                 //Schrittübersicht ausblenden
                 GameObject.Find("Panel2").GetComponent<CanvasGroup>().alpha = 0;
                 GameObject.Find("Panel2").GetComponent<CanvasGroup>().interactable = false;

                //Alle Schrittanleitungstexte ausblenden
                // anim = GameObject.Find("testmodell").GetComponent<Animation>();
                steps = GameObject.Find("Schritte");
                foreach(Text r in steps.GetComponentsInChildren<Text>()){
                        r.enabled = false;
                        }

                //Funktion increase_counter im Skript Schrittanl.cs ausführen (Blendet je
                nach aktuellem Schritt andere Texte und Animationen ein)
                GameObject.Find("MeineUI").GetComponent<schrittanl>().increase_counter();

                //Die 3 Komponenten, zu denen es eine Info-Box gibt, auf Default-Ebene
                setzen, damit sie klickbar sind
                foreach (Transform trans in obj.GetComponentsInChildren<Transform>(true)) {
        trans.gameObject.layer = LayerMask.NameToLayer("Default");
                }

                foreach (Transform trans in obj2.GetComponentsInChildren<Transform>(true)) {
        trans.gameObject.layer = LayerMask.NameToLayer("Default");
                }

                foreach (Transform trans in obj3.GetComponentsInChildren<Transform>(true)) {
        trans.gameObject.layer = LayerMask.NameToLayer("Default");
                }
    }

}
```

showtext.cs

Einfärben einer Komponente bei Berührung, Ein- und Ausblenden von UI-Elementen passend zur berührten Komponente

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class showtext : MonoBehaviour {

        public GameObject abc;
        public GameObject def;
        public GameObject hide1;
        public GameObject hide2;
        public GameObject hide3;
        public GameObject hide4;
        public GameObject hidecomp1;
        public GameObject hidecomp2;
        public GameObject heading;

        public Color originalColor1;
        public Color originalColor2;
        public Color myOriginalColor;

     void OnMouseDown()
     {
        /*Verstecke alle Textfelder und Buttons der anderen Komponenten*/
        hide1.SetActive(false);
        hide2.SetActive(false);
        hide3.SetActive(false);
        hide4.SetActive(false);

        /* Wenn Text und Button für die angetippte Komponente aktiv (=sichtbar) sind...*/
         if (abc.activeInHierarchy == true){
                /*...dann verstecke sie...*/
                abc.SetActive(false);
                def.SetActive(false);
                heading.SetActive(false);
                Debug.Log("Variable ist true");
                /* ...und färbe die Komponente in die Ausgangsfarbe */
                foreach(Renderer r in GetComponentsInChildren<Renderer>()){
                r.material.color= myOriginalColor;
                }
        }
        /* Wenn Text und Button für die Komponente nicht aktiv (=unsichtbar) sind...*/
        else{
                /* ...dann aktiviere sie...*/
                abc.SetActive(true);
                def.SetActive(true);
                heading.SetActive(true);
                Debug.Log("Variable ist false");
                /* ...färbe die Komponente rot...*/
                 foreach(Renderer r in GetComponentsInChildren<Renderer>()){
                r.material.color= Color.red;
                }
                /*...färbe die beiden anderen Komponenten in ihre Ausgangsfarbe zurück */
                foreach(Renderer r in hidecomp1.GetComponentsInChildren<Renderer>()){
                r.material.color= originalColor1;
                }
                foreach(Renderer r in hidecomp2.GetComponentsInChildren<Renderer>()){
                r.material.color= originalColor2;
                }

        }
        }
 }
```

Url.cs
Verlinkung zum CDP mit Vorauswahl der berührten Komponente, hier: Wassertank

```
using UnityEngine;

public class Url : MonoBehaviour {

        public void myurl ()
        {
        Application.OpenURL("http://www.technischeredaktion.com/icona/out/portal.htm?check18");
        }
}
```

UrlDuese.cs
Verlinkung zum CDP mit Vorauswahl der berührten Komponente, hier: Milchaufschäumdüse

```
using UnityEngine;

public class UrlDuese : MonoBehaviour {

        public void myurl ()
        {
        Application.OpenURL("http://www.technischeredaktion.com/icona/out/portal.htm?check10");
        }
}
```

UrlOverview.cs
Verlinkung zum CDP ohne Vorauswahl (von der Schrittübersicht aus)

```
using UnityEngine;

public class UrlOverview : MonoBehaviour {

        public void myurl ()
        {
        Application.OpenURL("http://www.technischeredaktion.com/icona/out/portal.htm");
        }
}
```

UrlSieb.cs
Verlinkung zum CDP mit Vorauswahl der berührten Komponente, hier: Siebhalter

```
using UnityEngine;

public class UrlSieb : MonoBehaviour {

        public void myurl ()
        {
        Application.OpenURL("http://www.technischeredaktion.com/icona/out/portal.htm?check14");
        }
}
```

6 Quellen

Microsoft (2016a): „using-Direktive (C#-Referenz)“. <https://msdn.microsoft.com/de-de/library/sf0df423.aspx> [Stand: 2016. Zugriff: 04.07.2016 11:27 MESZ]

Microsoft (2016b): „Zugriffsmodifizierer (C#-Programmierhandbuch)“ <https://msdn.microsoft.com/de-de/library/ms173121.aspx> [Stand: 2016. Zugriff: 04.07.2016 11:35 MESZ]

PTC Inc. (2016): „Vuforia Supported Versions | Vuforia Library Prod“. <https://developer.vuforia.com/library/articles/Solution/Vuforia-Supported-Versions> [Stand: 2016. Zugriff: 30.06.2016 10:02 MESZ]

Roden, Golo (2010): „guide to C# | Guide | Einführung in .NET“. <http://www.guidetocsharp.de/Introduction.aspx> [Stand: 20.05.2010. Zugriff: 30.06.2016 11:29 MESZ]

Unity Technologies (2016): „Unity - Manual: Designing UI for Multiple Resolutions“. <https://docs.unity3d.com/Manual/HOWTO-UIMultiResolution.html> [Stand: 2016. Zugriff: 02.07.2016 18:30 MESZ]

YouTube (2013): „Creating Custom Pivots in Unity – YouTube”. <https://www.youtube.com/watch?v=itIh8PYGP7w> [Stand: 14.05.2013. Zugriff: 03.07.2016 17:05 MESZ]

YouTube (2015): „[iPhone] AR(virtual button) sample with Vuforia SDK – YouTube”. <https://i.ytimg.com/vi/Ji0wUmR5dt8/maxresdefault.jpg> [Stand: 29.06.2015. Zugriff: 03.07. 2016 12:22 MESZ]