Landesinstitut für Schulentwicklung

Thema:	**App-Entwicklung mit Android Studio**
Name der Autorin/ des Autors:	Jochen Pogrzeba,StR Max-Weber-Schule, Freiburg
Fach:	Informatik, Wirtschaftsinformatik
Klasse/Jahrgangsstufe:	JG1
Schulart:	Berufliches Gymnasium
Lehrplanbezug:	LPE 9: Objektorientierte Systementwicklung (Fortführung) LPE 9: Objektorientierte Systemanalyse und -entwicklung
Zeitumfang:	-
Betriebssystem/e:	Android
Apps:	-
Technische Settings:	Schüler-PC's mit Windows und Android Studio Ggf. Schüler-Tablets mit Android

Fachliche Ziele:

- Kenntnisse der Besonderheiten der App-Entwicklung.
- Umsetzung des ereignisorientierten Ansatzes der GUI-Entwicklung.
- Vertiefung der Kenntnisse in einer objektorientierten Programmiersprache.
- Vertiefung der Kenntnisse in einer strukturierenden Auszeichnungssprache.

Sonstige Ziele:

- Erweiterung der Sozialkompetenz durch zusammenführende Gruppen- oder Projektarbeit.
- Schulung der gestalterischen Kreativität.
- Förderung der Medienkompetenz.
- Evtl. Schulung der Präsentationskompetenz.
- Evtl. individuelle Förderung.

Diese Unterrichtseinheit ist nicht als direkt umsetzbares Material konzipiert. Vielmehr soll dieses Beispiel der Lehrerin/dem Lehrer die Möglichkeit geben, sich ggf. schnell in das Thema einzuarbeiten. Die direkte Umsetzung des Beispiels mit Schülern ist möglich, aber nicht Hauptziel. Diese Unterrichtseinheit soll den Lehrer motivieren, ein eigenes, evtl. profilbezogenes Beispiel, mit den Schülern umzusetzen.
Aus diesem Grunde wurde darauf verzichtet einen Verlaufsplan hinzuzufügen.

Landesinstitut für Schulentwicklung

App-Entwicklung mit Android Studio
- Entwicklung einer Memory-App –
(Ein Beispiel für Fortgeschrittene)

Autor: Jochen Pogrzeba, StR
Max-Weber-Schule, Freiburg

Inhaltsverzeichnis:

jochen.pogrzeba@max-weber-schule.de

1 Vorbemerkungen

In dieser Unterrichtseinheit wird mit Hilfe der Entwicklungsumgebung *Android Studio* ein Memory-Spiel entwickelt, das auf Android-Tablets lauffähig ist.

Als Programmieroberfläche wurde für dieses Unterrichtsbeispiel *Android Studio* benutzt, da diese Software von Google als offizielle IDE zur App-Entwicklung für Android unterstützt wird. Eclipse mit seinen Plugins wird von Google nicht mehr unterstützt.
Android Studio basiert auf der IDE *IntelliJ IDEA*, welche sich ähnlich wie Eclipse bedienen lässt, der Einarbeitungsaufwand für Lehrer und Schüler ist daher sehr überschaubar.

Als Programmierbeispiel wurde ein Memory-Spiel gewählt. Es enthält keine typischen Tablet-Funktionalitäten wie Scrollen oder Kippen. Der Fokus liegt auf den hierfür verwendeten Programmstrukturen. (Für ein komplexeres Projekt siehe die Unterrichtseinheit *UE Taschenrechner-App).*

Hauptaugenmerk dieser Beschreibung liegt in der Vorstellung der grundsätzlichen Prinzipien der Android-Entwicklung. Hauptziel dieser Unterrichtseinheit ist es, der Lehrerin/dem Lehrer eine Anleitung mitzugeben, mit der er sich leicht und schnell in die Thematik einarbeiten kann. Eine direkte Umsetzung des vorgestellten Projektes im Unterricht ist zwar möglich, aber nicht primäres Ziel. Der Leser ist also eingeladen, eine Unterrichtseinheit mit einer eigenen Softwareidee durchführen.

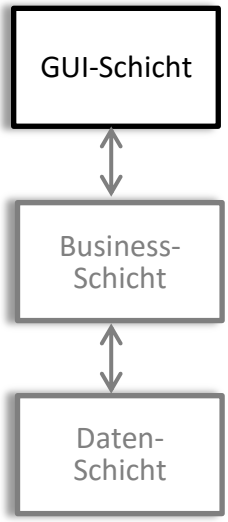
Das zugehörige Android Studio-Projekt finden Sie im Archiv *Projekt Memory.*

Bei der Lektüre dieser Unterrichtseinheit wird folgendes vorausgesetzt:
- Ein installiertes und eingerichtetes Android Studio ist verfügbar (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Die Grundstruktur eines Android-Projektes ist bekannt (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Kenntnisse in Java und idealerweise Swing.

Sinnvoll wären Kenntnisse in:
- Umgang mit einer IDE wie z.B. Eclipse.
- XML

2 Beschreibung der GUI-Schicht

activity_main.xml
MainActivity.java



Ein- und Ausgabe der Werte auf der Benutzeroberfläche und Weiterleitung an die Business-Schicht

Nicht vorhanden, da dieses Projekt nur die Funktionalität auf der GUI zeigen soll.

Nicht vorhanden, da eine persistente Datenhaltung hier nicht erforderlich ist.

Die beiden zentralen Dateien der GUI-Schicht sind:

Activity_main.xml

In dieser XML-Datei werden die GUI-Elemente als XML-Tags eingetragen. Auch die Attribute und deren Werte finden sich hier wieder. Diese Datei ist somit der zentrale Ort, der das Layout der App festlegt. Diese Datei wird beim Start automatisch erzeugt. Die Einträge für die GUI-Elemente werden automatisch hinzugefügt, wenn man im Designmodus die GUI-Objekte auf die Activity zieht. Trotzdem kann es nötig sein, bei komplexeren Projekten auch direkt in dieser Datei zu arbeiten.

Im Projektbaum finden sich zwei *Activity_main.xml*-Dateien:
Mit Zusatz *(land)*: Hier wird das Design für die Horizontalansicht festgelegt
Ohne Zusatz: Hier wird das Design für die Vertikalansicht festgelegt.

Dies bedeutet, dass man für beide Bildschirmausrichtungen völlig unterschiedliche Designs festlegen kann. Die App kann also auf das Kippen des Tablets reagieren. Für dieses einfache Projekt wird nur die Vertikalansicht benutzt.

MainActivity.java

Diese Java-Klasse repräsentiert die MainActivity, die unsere App steuert. Sie ist somit die zentrale Datei, was die Steuerung der App über die GUI-Elemente angeht. Sie erbt Funktionalität von anderen Activity-Klassen, je nachdem was bei der Projekterstellung als voreingestellte Activity ausgewählt wurde.

Diese Klasse besitzt die Methode *onCreate()*, die beim Starten der App ausgeführt wird. In ihr wird zum einen ein Exemplar der Superklasse erzeugt, zum anderen die *Activity_main.xml* als Layout-Datei eingefügt. Diese wird im weiteren Verlauf durch die Klasse *R* repräsentiert (siehe Kap. 4).

Ähnlich einer GUI-Klasse bei der Swing-Programmierung werden hier auch die GUI-Objekte deklariert und deren Eventhandler hinzugefügt.

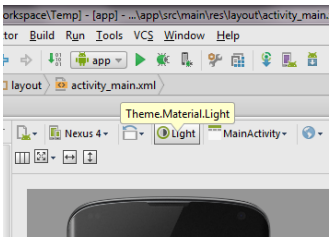
3 Erstellung der Benutzeroberfläche

3.1 Layout

Eine Activity wie diese kann nahezu vollständig mittels Drag-and-Drop im Designmodus erstellt werden. Dabei wird dann im Hintergrund die *activity_main.xml* mit den entsprechenden Tags gefüllt.
Bei komplexen Projekten wird man aber nicht darum herumkommen direkt in dieser Datei zu arbeiten, da bei GUIs mit vielen GUI-Objekten und Scroll-Bereichen die Handhabung im Design-Modus sehr unübersichtlich werden kann.

Bei einem neu erstellten Projekt sind bereits ein Layout und ein Textfeld mit dem Inhalt "Hello world" vorbereitet.

Hinweis: Sollte der Designmodus hier "Rendering Problems" anzeigen, so empfiehlt es sich das Grafikthema zu ändern. Ändern Sie in diesem Falle *ProjectTheme* auf ein anderes Thema, z.B. *MaterialLight*.



Bevor die Elemente auf der Activity platziert werden können, muss ein Layout platziert werden. Ein Layout kann man sich wie einen Teppichboden vorstellen, auf dem Möbelstücke nach bestimmten Mustern angeordnet werden. Das voreingestellte Layout ist das *RelativeLayout*, hier werden die GUI-Objekte relativ zu der Position der anderen Elemente platziert (ähnlich wie HTML ohne css).

Für komplexere Projekte empfiehlt sich ein leichter handhabbares Layout wie das *GridLayout*, hier wird allerdings das *RelativeLayout* beibehalten.

In der *activity_main.xml* wird nun das *RelativeLayout* als XML-Tag angezeigt. Zwischen `<RelativeLayout… >` und `</RelativeLayout>` werden dann später die XML-Tags für die anderen GUI-Objekte stehen.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">
```

Ein GUI-Objekt lässt sich nun im Designmodus per Drag-und-Drop in das *RelativeLayout* platzieren.

3.2 Platzierung der GUI-Objekte

Auf die Activity werden nun zwölf ImageButtons in drei Spalten gesetzt.

```
<ImageButton
    android:layout_width="300px"
    android:layout_height="300px"
    android:id="@+id/img1"
    android:layout_row="0"
    android:layout_column="0"
    android:clickable="true"
    android:src="@drawable/wolken" />
<ImageButton
    android:layout_width="300px"
    android:layout_height="300px"
    android:id="@+id/img2"
    android:layout_row="0"
    android:layout_column="1"
    android:clickable="true"
    android:src="@drawable/wolken" />
<ImageButton
    android:layout_width="300px"
```

Über den Parameter src lässt sich das Bild auswählen, welches auf dem Button erscheinen soll. Dies ist für alle zu Beginn das Wolkenbild. Die Bilder müssen im Projektverzeichnis unter *res\drawable* abgelegt sein.

Wichtig ist vor allem der Name des Objektes. Vergeben Sie bei *id* (im Properties-Bereich) oder `android:id="@+id/txtNameDesObjektes"` (in der *activity_main.xml)* einen aussagekräftigen Namen.
z.B. *img1, img2* usw.

4 Programmierung in der MainActivity.java

Diese Java-Klasse repräsentiert die MainActivity, die unsere App steuert. Sie ist somit die zentrale Datei, was die Steuerung der App über die GUI-Elemente angeht. Sie wird beim Erstellen des Projektes mit einigen vorbereiteten Methoden bereitgestellt.

4.1 Die MainActivity im Ausgangszustand

Die Klasse *MainActivity* erbt von der Klasse *AppCompatActivity* (bzw. bei älteren Versionen von *ActionBarActivity)*.
Die Methoden *onCreateOptionsMenu()* und *onOptionsItemSelected()* werden nur benötigt, wenn die App ein Menü bekommen soll und werden deshalb hier nicht weiter behandelt.

Die Methode *onCreate()* wird ausgeführt, wenn die App ein Exemplar der *MainActivity* erstellt, soll heißen, wenn die App gestartet wird. Sie führt folgendes durch:

- Erstellung eines Exemplars der Superklasse.
- Der Klasse *R* wird die Datei *activity_main.xml* als Attribut hinzugefügt. Die Klasse *R* repräsentiert somit die dort erstellte GUI. Die GUI-Objekte wie Buttons, Textfelder usw. sind somit Attribute dieser Klasse *R* und können nun in der *MainActivity* verwendet werden.
- Die GUI-Objekte wie Buttons, Textfelder usw. werden registriert und mit einem Eventhandler ausgestattet der Ereignisse wie Klicken o.ä. verarbeitet.

Letztere Aufgaben werden hier in die Methode *bilderRegistrieren()* ausgelagert.

Der Methode *onCreate()* wird beim Start vom System ein Bundle-Objekt mitgegeben. In ihm können Zustände einer App gespeichert werden, was aber hier nicht benötigt wird.

4.2 Die MainActivity für die Memory-App

Wir haben der Klasse mehrere Attribute und zwei weitere Methoden spendiert. (siehe Erklärung des Quellcodes unten)

Die neue Funktionalität wird der Methode *onCreate()* hinzugefügt. Die Methode *onCreate()*: (Die neuen Inhalte sind fett gedruckt)

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.bilderRegistrieren();
}
```

Die Registrierung der benötigten GUI-Objekte (d.h. die zwölf ImageButtons) wird in der Methode *bilderRegistrieren()* durchgeführt.

Registrierung der GUI-Objekte:
Beispiel für den ersten ImageButton, die anderen laufen analog:

```
final ImageButton img1 = (ImageButton) findViewById(R.id.img1);
img1.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        umdrehen(1,img1,R.drawable.mannheim);
    }
});
```

Hier werden die GUI-Objekte mit Hilfe der Klasse *R* erstellt. Ähnlich wie bei JavaScript werden die in der Klasse *R* repräsentierten GUI-Elemente über eine Methode *findViewById* als GUI-Objekt gespeichert. Ein Cast übernimmt die Umwandlung der allgemeinen GUI-Elemente in ein spezielles GUI-Objekt.

ActionListener für den Button *img1:*
Ein ActionListener "horcht", ob ein Ereignis auf dem Button ausgeführt wird, z.B. ein Klick. Ist dem so, wird der Inhalt der Methode *onClick()* ausgeführt. *onClick()* ist dabei die von uns fertig programmierte Methode eines Interface.

Aufruf der Methode *umdrehen*:
Beim Klick auf einen ImageButton wird die Methode *umdrehen()* aufgerufen. Dieser werden folgende Parameter übergeben:

- die Bildnummer (Beim ersten Bild die 1, beim zweiten die 2 usw.)
- Das ImageButton-Objekt (hier *img1*)

- Die Resource, d.h. das Bild welches „hinter“ dem noch zugedeckten Bild liegt. (hier: *R.drawable.mannheim*) Somit wird an dieser Stelle festgelegt, welches Motiv sich hinter welchen ImageButton befinden soll.

4.3 Methode umdrehen()

Im Folgenden wird die Methode *umdrehen()* schrittweise erläutert. Jeder Zwischenschritt ist dabei einzeln ausführbar, um den Fortschritt der App sichtbar zu machen.
Diese Methode leistet die eigentliche Funktionalität: „Umdrehen“ der Bilder, Vergleich und Auswertung. Sie befindet sich in der Klasse *MainActivity*.

Die Grundidee:

Schritt 1: Einfügen des Arrays offen[] und ‚Umdrehen‘ des Bildes

Idee: In dieser Ausbaustufe sorgen wir dafür, dass der offen-Status geändert wird, wenn man auf ein Bild klickt, sowie das Motiv geändert wird (Eine Abfrage, ob die Treffer gleich sind gibt es hier noch nicht)

```
private boolean offen[] = new boolean[13];
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (offen[bildnr])
      {
            bild.setImageResource(R.drawable.wolken);
            offen[bildnr] = false;
```

Attribute der Klasse MainActivity.

Wenn das aktuelle Bild aufgedeckt ist, dann wird dem Bild das Wolkenmotiv gegeben (d.h. es wird zugedeckt). Außerdem wird der offen-Status auf ‚false‘ gesetzt. Damit wurde das Bild ‚umgedreht‘ und entsprechend gekennzeichnet.

```
        }
        else
        {
                bild.setImageResource(bildname);
                offen[bildnr] = true;
        }
}
```

Wenn aber das aktuelle Bild zugedeckt ist, wird dem Bild das Motiv gegeben, welches dem Bildfeld in *bilderRegistrieren()* zugeordnet wurde. Der offen-Status wird auf ‚true' gesetzt.

Wenn man hier draufklickt, geht das Programm in den **else**-Zweig

Wenn man hier draufklickt, geht das Programm in den **if**-Zweig

Schritt 2: Nur zwei Bilder sollen zu sehen sein

Idee: In dieser Ausbaustufe sorgen wir dafür, dass es das Programm erfährt, wenn ein zweites Bild angeklickt wurde. Danach soll das zuletzt angeklickte Bild wieder umgedreht werden.

```
private boolean offen[] = new boolean[13];
private int anzahlOffen=0;
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (offen[bildnr])
      {
            bild.setImageResource(R.drawable.wolken);
            offen[bildnr] = false;
            this.anzahlOffen--;
      }
      else
      {
            bild.setImageResource(bildname);
            offen[bildnr] = true;
            this.anzahlOffen++;
      }

      if (anzahlOffen == 2)
      {
```

Die Anzahl geöffneter Bilder ist standardmäßig 0

Wurde ein Bild zugedeckt, wird die Anzahl offener Bilder um 1 verringert.

Wurde ein Bild aufgedeckt, wird die Anzahl offener Bilder um 1 erhöht.

Ist das angeklickte Bild das zweite Geöffnete?

```
            anzahlOffen--;
            geloest[bildnr]=true;
            letztesBild.setImageResource(R.drawable.wolken);
      }
}
```

Falls ja:
- zeige wieder das Wolkenbild
- setze den offen-Status auf false
- Es ist nun ein Bild weniger umgedreht, deswegen ziehe von anzahlOffen den Wert 1 ab.

Schritt 3: Abgleich, ob Treffer oder nicht

Idee: In dieser Ausbaustufe sorgen wir dafür, dass das Programm überprüft, ob die zwei geöffneten Bilder gleich sind.

```
private boolean offen[] = new boolean[13];
private int anzahlOffen=0;
private ImageButton letztesBild;
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (offen[bildnr])
      {
            bild.setImageResource(R.drawable.wolken);
            offen[bildnr] = false;
            this.anzahlOffen--;
      }
      else
      {
            bild.setImageResource(bildname);
            bild.setTag(bildname);
            offen[bildnr] = true;
            this.anzahlOffen++;
            if (anzahlOffen == 1)
            {
                  letztesBild = bild;
            }
      }
```

Man benötigt eine neue Variable in der der Name des zuerst angeklickten Bildes gespeichert wird

Das Attribut *tag* des Bild-Objektes wird mit dem Namen des aktuellen Bildes gefüllt. Dies wird zum Trefferabgleich benötigt.

Ist durch den aktuellen Mausklick gerade das erste Bild geöffnet worden?
Falls ja: Speichere das "dahinterliegenden" Bild als letztesBild

```
      if (anzahlOffen == 2)
      {
            if (bild.getTag().toString().equals(letztesBild.getTag().toString()))
            {
                  Toast.makeText(this, "Treffer!", Toast.LENGTH_SHORT).show();
            }
            else
            {
                  Toast.makeText(this, "Falsch", Toast.LENGTH_SHORT).show();
                  bild.setImageResource(R.drawable.wolken);
                  offen[bildnr] = false;
                  anzahlOffen--;
            }
      }
}
```

Wenn die Tags der beiden Bildobjekte übereinstimmen…
Oder anders gefragt: Haben wir jetzt und ein Klick davor zwei Bilder mit gleichem Namen angeklickt? Dann war es wohl ein Treffer!

Bsp.: Beim ersten Mausklick wird Ulm umgedreht und 'ulm' in letztesBild gespeichert.
Beim zweiten Mausklick wird der Schwarzwald umgedreht und geprüft, ob 'ulm' = 'schwarzwald' → kein Treffer, Ulm wird wieder umgedreht.

Bsp.: Beim ersten Mausklick wird Ulm umgedreht und 'ulm' in letztesBild gespeichert.
Beim zweiten Mausklick wird Ulm umgedreht und geprüft, ob 'ulm' = 'ulm' → Treffer!
Weiter passiert (erstmal) nichts.

Schritt 4: Falsche Bilder wieder umdrehen

Idee: In dieser Ausbaustufe sorgen wir dafür, dass beide Bilder wieder umgedreht werden, wenn sie nicht übereinstimmen. In der letzten Ausbaustufe wurde immer nur das gerade angeklickte Bild wieder umgedreht.

```
private boolean offen[] = new boolean[13];
private int anzahlOffen=0;
private ImageButton letztesBild;
private int letztesBildNr;
…
```

Zusätzlich zum Namen des letzten Bildes müssen wir nun auch die Bildnummer des letzten Bildes speichern.
Grund: Man muss beim zweiten Mausklick den offen-Status und das Motiv des vorher geklickten Bildes ändern können. Da kommt man nur über die bildnr ran

```
public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (offen[bildnr])
      {
            bild.setImageResource(R.drawable.wolken);
            offen[bildnr] = false;
            this.anzahlOffen--;
      }
      else
      {
            bild.setImageResource(bildname);
            bild.setTag(bildname);
            offen[bildnr] = true;
            this.anzahlOffen++;
            if (anzahlOffen == 1)
            {
                  letztesBild = bild;
                  letztesBildNr = bildnr;
            }
```

Ist durch den aktuellen Mausklick gerade das erste Bild geöffnet worden?
Falls ja: Speichere nun auch noch die aktuelle *bildnr* als *letztesBildnr*

```
        }
        if (anzahlOffen == 2)
        {
              if (bild.getTag().toString().equals(letztesBild.getTag().toString()))
              {
                    Toast.makeText(this, "Treffer!"
                    anzahlOffen--;
                    anzahlOffen--;
              }
              else
              {
                    Toast.makeText(this, "Falsch", Toast.LENGTH_SHORT).show();
                    bild.setImageResource(R.drawable.wolken);
                    letztesBild.setImageResource(R.drawable.wolken);
                    offen[bildnr] = false;
                    offen[letztesBildNr] = false;
                    anzahlOffen--;
                    anzahlOffen--;
              }
        }
}
```

War es ein Treffer? Falls ja, lass die Bilder so stehen, aber zieh 2 von anzahlOffen ab.
(Die geöffneten Bilder sind aus dem Spiel, die interessieren jetzt nicht mehr)

War es kein Treffer?
Zeige nun auch dort Wolken, wo der erste Mausklick war, setze dort den offen-Status als false und ziehe noch 1 von anzahlOffen ab

Schritt 5: Umgedrehte Bilder sind nicht mehr anklickbar

Idee: In dieser Ausbaustufe sorgen wir dafür, man bereits umgedrehte Bilder nicht mehr anklicken kann (oder besser gesagt: es soll nichts passieren, wenn man sie anklickt).

```
private boolean offen[] = new boolean[13];
private boolean geloest[] = new boolean[13];
private int anzahlOffen=0;
private ImageButton letztesBild;
private int letztesBildNr;
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (!geloest[bildnr])
      {
            if (offen[bildnr])
            {
                  bild.setImageResource(R.drawable.wolken);
                  offen[bildnr] = false;
                  this.anzahlOffen--;
            }
            else
            {
                  bild.setImageResource(bildname);
                  bild.setTag(bildname);
                  offen[bildnr] = true;
                  this.anzahlOffen++;
                  if (anzahlOffen == 1)
                  {
                        letztesBild = bild;
                        letztesBildNr = bildnr;
                  }
            }
```

Ein neues Array!
Jedes Bild bekommt noch einen neuen Status, nämlich ob es gelöst ist (true) oder nicht (false).

Wenn der geloest-Status des gerade geklickten Bildes nicht true ist, dann führe alles wie bisher aus.

Ansonsten lass das Bild unberührt.

```
            if (anzahlOffen == 2)
            {
                  if (bild.getTag().toString().equals(letztesBild.getTag().toString()))
                  {
                        Toast.makeText(this, "Treffer!", Toast.LENGTH_SHORT).show();
                        anzahlOffen--;
                        anzahlOffen--;
                        geloest[bildnr]=true;
                        geloest[letztesBildNr]=true;
                  }
                  else
                  {
                      Toast.makeText(this, "Falsch", Toast.LENGTH_SHORT).show();
                      bild.setImageResource(R.drawable.wolken);
                      letztesBild.setImageResource(R.drawable.wolken);
                      offen[bildnr] = false;
                      offen[letztesBildNr] = false;
                      anzahlOffen--;
                      anzahlOffen--;
                  }
             }
      }
}
```

Wenn der zweite Klick den Treffer gebracht hat, setze beide Bilder als gelöst.

Das Ulmer Münster war bereits umgedreht. Die beiden (nichtpassenden) Bilder vom aktuellen Klickvorgang werden als nicht gelöst gesetzt.

Schritt 6: (Meldung, wenn fertig)

Idee: In dieser Ausbaustufe sorgen wir dafür, dass eine "Fertig!"-Meldung erscheint, wenn alle Bilder umgedreht sind.

```
private boolean offen[] = new boolean[13];
private boolean geloest[] = new boolean[13];
private int anzahlOffen=0;
private int anzahlGeloest=0;
private ImageButton letztesBild;
```

Wir führen eine neue Variable ein, mit der wir zählen, wie viele Bilder gelöst sind. Zu Beginn sind es natürlich null.
Diese Variable muss außerhalb der Funktion stehen, da sie sonst bei jedem Klick wieder auf null gesetzt werden würde. So passiert das nur ganz zu Beginn.

```
private int letztesBildNr;
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (!geloest[bildnr])
      {
            if (offen[bildnr])
            {
                  bild.setImageResource(R.drawable.wolken);
                  offen[bildnr] = false;
                  this.anzahlOffen--;
            }
            else
            {
                  bild.setImageResource(bildname);
                  bild.setTag(bildname);
                  offen[bildnr] = true;
                  this.anzahlOffen++;
                  if (anzahlOffen == 1)
                  {
                        letztesBild = bild;
                        letztesBildNr = bildnr;
                  }
            }
            if (anzahlOffen == 2)
            {
                  if (bild.getTag().toString().equals(letztesBild.getTag().toString()))
                  {
                        Toast.makeText(this, "Treffer!", Toast.LENGTH_SHORT).show();
                        anzahlOffen--;
                        anzahlOffen--;
                        anzahlGeloest++;
                        anzahlGeloest++;
                        geloest[bildnr]=true;
                        geloest[letztesBildNr]=true;
                  }
                  else
                  {
                      Toast.makeText(this, "Falsch", Toast.LENGTH_SHORT).show();
                      bild.setImageResource(R.drawable.wolken);
                      letztesBild.setImageResource(R.drawable.wolken);
                      offen[bildnr] = false;
                      offen[letztesBildNr] = false;
                      anzahlOffen--;
                      anzahlOffen--;
                  }
            }
            if (anzahlGeloest==12)
            {
                  Toast.makeText(this, "Fertig", Toast.LENGTH_SHORT).show();
            }
      }
}
```

Bei einem Treffer soll die Variable anzahlGeloest um zwei hochgezählt werden.

Ist die Variable bei 12 angekommen, sind alle Bilder gelöst worden.

Schritt 8: Anzeige wie viel Mausklicks benötigt wurden

Idee: In dieser (letzten) Ausbaustufe sorgen wir dafür, dass in der "Fertig!"-Meldung angezeigt wird, wie viel Mausklicks man zur Lösung benötigt hat. Dies ist der letzte Schritt, d.h der endgültige Code in der Klasse MainActivity lautet wir folgt:

```
private boolean offen[] = new boolean[13];
private boolean geloest[] = new boolean[13];
private int anzahlOffen=0;
private int anzahlGeloest=0;
private ImageButton letztesBild;
private int letztesBildNr;
private int klicks;
…

public void umdrehen(int bildnr,ImageButton bild,int bildname)
{
      if (!geloest[bildnr])
      {
            klicks++;
            if (offen[bildnr])
            {
                  bild.setImageResource(R.drawable.wolken);
                  offen[bildnr] = false;
                  this.anzahlOffen--;
            }
            else
            {
                  bild.setImageResource(bildname
                  bild.setTag(bildname);
                  offen[bildnr] = true;
                  this.anzahlOffen++;
                  if (anzahlOffen == 1)
                  {
                        letztesBild = bild;
                        letztesBildNr = bildnr;
                  }
            }
            if (anzahlOffen == 2)
            {
                  if (bild.getTag().toString().equals(letztesBild.getTag().toString()))
                  {
                        Toast.makeText(this, "Treffer!", Toast.LENGTH_SHORT).show();
                        anzahlOffen--;
                        anzahlOffen--;
                        anzahlGeloest++;
                        anzahlGeloest++;
                        geloest[bildnr]=true;
                        geloest[letztesBildNr]=true;
                  }
                  else
                  {
                      Toast.makeText(this, "Falsch", Toast.LENGTH_SHORT).show();
                      bild.setImageResource(R.drawable.wolken);
                      letztesBild.setImageResource(R.drawable.wolken);
                      offen[bildnr] = false;
                      offen[letztesBildNr] = false;
                      anzahlOffen--;
                      anzahlOffen--;
                  }
            }
```

Wir führen eine neue Variable ein, mit der wir zählen, wie viele Mausklicks der User getätigt hat. Diese Variable muss außerhalb der Funktion stehen, da sie sonst bei jedem Klick wieder auf null gesetzt werden würde. So passiert das nur ganz zu Beginn.

Immer wenn die Methode durchlaufen wird, wird die Variable *klicks* um 1 erhöht.
Wenn die Anweisung hier steht, wird jeder Mausklick gezählt, außer man klickt auf ein bereits gelöstes Bild.

```
            if (anzahlGeloest==12)
            {
                  Toast.makeText(this, "Fertig. Sie haben "+klicks+" Klicks
                        benötigt!", Toast.LENGTH_SHORT).show();
            }
      }
}
```

5 Weitere Vorgehensweise

Sie können nun die App in Ihrer Testumgebung testen. Siehe dazu *Handreichung - Installation und Einrichtung Android Studio*.