Globetrotter – Dokumentation

Gümüs Ali

Mayer Stefan

Moosbrugger Simon

Spiegel Philipp

Inhalt

[1. Thema 3](#_Toc377739574)

[1.1. Neuausrichtung nach ÖkoSwim 3](#_Toc377739575)

[1.1.1. Warum nicht mehr ÖkoSwim 3](#_Toc377739576)

[1.1.2. Analyse anderer Open Data Datenquellen 3](#_Toc377739577)

[1.2. Globetrotter 4](#_Toc377739578)

[2. Analyse - Technologien 5](#_Toc377739579)

[2.1. Applikation 5](#_Toc377739580)

[2.1.1. HTML 5 mit WebGL 5](#_Toc377739581)

[2.1.2. Blender 5](#_Toc377739582)

[2.1.3. Unity3D 6](#_Toc377739583)

[2.2. Steuerung 6](#_Toc377739584)

[2.2.1. Android Smartphone 6](#_Toc377739585)

[2.2.2. iOS Smartphone 6](#_Toc377739586)

[2.2.3. Einsatz von Bewegungssensoren und Kameras 7](#_Toc377739587)

[3. Paperprototypes und Interaktionen 8](#_Toc377739588)

[3.1. Erste Skizzen 8](#_Toc377739589)

[3.1.1. Auswahl von Kontinenten, Ländern und Indikatoren 8](#_Toc377739590)

[3.1.2. Vergleich von Ländern 9](#_Toc377739591)

[3.2. Skizzen zum User Interface 10](#_Toc377739592)

[3.2.1. Länderauswahl 10](#_Toc377739593)

[3.2.2. Vergleich von Ländern 11](#_Toc377739594)

[3.3. Interaktionen 12](#_Toc377739595)

[3.3.1. Länderauswahl 12](#_Toc377739596)

[3.3.2. Datenvisualisierung 15](#_Toc377739597)

[4. Realisierung 16](#_Toc377739598)

[4.1 Verwendete Tools 16](#_Toc377739599)

[4.2 Unity-Anwendung 16](#_Toc377739600)

[4.2.1 Ländererkennung und Koordinatenumrechnung 16](#_Toc377739601)

[4.3 Android-App 20](#_Toc377739602)

[5. Probleme während der Entwicklung und deren Lösung 21](#_Toc377739603)

[5.1. Anfrage an den Webservice der Weltbank 21](#_Toc377739604)

[5.2. Zufälligerweise 502 Bad Gateway 21](#_Toc377739605)

[5.3. Textencoding in Unity 22](#_Toc377739606)

[5.4. Smartphone, Datenübertragung 23](#_Toc377739607)

[5. Systemanforderungen 24](#_Toc377739608)

# 1. Thema

## 1.1. Neuausrichtung nach ÖkoSwim

### 1.1.1. Warum nicht mehr ÖkoSwim

Im Laufe des Projekts stellten wir fest, dass die gelieferten Daten vom Land Oberösterreich teilweise veraltet sind. Die Daten sind auch nicht vollständig sondern nur sporadisch erfasst worden. Außerdem sind die Datenformate auch nicht immer gut maschinenlesbar.

Wir betrachteten natürlich auch andere Open Data Datensätze vom Land Oberösterreich, ob eine andere Möglichkeit für eine neue Entwicklung besteht. Aber es gibt schon eine Applikation, die beinahe alle Daten verwertet. Da somit kaum Raum für eine neuartige Applikation, die einen Nutzen für die Allgemeinheit hat, besteht, haben wir uns im Rahmen des Coachings für eine komplette Neuausrichtung entschieden.

### 1.1.2. Analyse anderer Open Data Datenquellen

Während der Neuausrichtungsphase betrachteten wir unterschiedliche Open Data Datenquellen genauer. Hier ist eine Auswahl davon mit ihren Vor- und Nachteilen:

|  |  |
| --- | --- |
| Land Vorarlberg | - geringe Auswahl an Datensätzen  - Datensätze enthalten nicht viel Informationen  - alle Formate fast nicht maschinenlesbar |
| Stadt Wien | + oft gut maschinenlesbar  + Daten werden aktualisiert  - viele Applikationen bereits umgesetzt |
| Schweiz | + viele Daten  + Aktualität OK  - Datenformat mehrmals nur schwer maschinenlesbar |
| Weltbank | + viele verschiedene Datensätze  + gute Aktualität  + REST Webservice zur Datenabfrage  - manchmal fehlen erfasste Länder oder Jahre |

## 1.2. Globetrotter

Wegen der Fülle an Datensätzen und derer hohen Vollständigkeit und Aktualität, konsumieren wir nun die Daten von der Weltbank (<http://data.worldbank.org>). Das neue Projekt *Globetrotter* dient dazu, auf einem Globus Länder auszuwählen und diese in unterschiedlichen Datensätzen zu vergleichen.

Das Ziel der Applikation ist, dass man die Welt auf einem Globus erkunden kann. Dabei soll grundlegendes geographisches Wissen sowie Detailwissen zu den Datenkategorien der Weltbank aufgebaut werden können. Als Einsatzgebiet sehen wir vor allem den Bereich des E-Learning vor.

# 2. Analyse - Technologien

## 2.1. Applikation

Für die Applikation werden 3D-Fähigkeiten benötigt. Folgende Technologien beziehungsweise Game Engines stehen unter anderen zur Auswahl:

* WebGL
* Blender
* Unity3D

### 2.1.1. HTML 5 mit WebGL

Die Recommendations von HTML 5 und WebGL stehen frei und plattformunabhängig zur Verfügung. Da sie aber noch relativ neu sind, können mehrere Schwierigkeiten auftreten. Nicht alle Webbrowser unterstützen HTML und WebGL im nötigen Ausmaß (Microsoft Internet Explorer kann kein WebGL; Apple Safari muss WebGL zuerst umständlich aktivieren). Zudem muss eine aktuellere Grafikkarte mit aktuellem Grafiktreiber am System installiert sein. Unter diesen Umständen würden viele potentielle User die Applikation nicht nutzen können. Die Teammitglieder haben auch kaum Erfahrung mit diesen Technologien. Zudem scheint die Performance nach eigenen Tests noch nicht ganz ausgereift zu sein.

### 2.1.2. Blender

Blender steht für verschiedene Plattformen (Linux, Mac, Windows) frei zur Verfügung. Es enthält neben der 3D-Modellierung und –Animation die sogenannte Blender Game Engine. Für die Programmierung stellt die Blender Game Engine eine Python API zur Verfügung. Da wir im Team aber keine Erfahrungen mit der Blender Game Engine und Python haben, haben wir uns gegen die Blender Game Engine entschieden.

### 2.1.3. Unity3D

Unity3D ist eine Game Engine. Neben der kostenpflichtigen Version Unity3D Pro gibt es die kostenfreie Version Unity3D Free. Seit der Programmversion 4.x ist für Unity3D Free lediglich ein kostenfreier Account beim Hersteller nötig. Mit Mac und Windows werden auch die zwei großen Plattformen unterstützt. Bei der Programmierung in Unity3D kann zwischen Boo, C# und JavaScript gewählt werden. Da Unity3D eine starke Verbreitung in der einschlägigen Branche aber auch darüber hinaus hat, bietet es sich an. Außerdem ist allen Teammitglieder die Programmiersprache C# gut bekannt. Der Umfang der Dokumentation und eine aktive Community sorgen für viele Hilfestellungen. Leider fehlen die wichtigen Teile der .NET API zur Erzeugung und Bearbeitung der Grafik. Deshalb müssen die Diagramme per Komposition und Manipulation von 3D-Objekten erstellt werden. Dennoch haben wir und entschlossen, dass Unity3D für die Applikation zum Einsatz kommen soll.

## 2.2. Steuerung

Die Steuerung der Applikation soll nicht direkt erfolgen. Stattdessen ist geplant, dass der User die Applikation über ein eigenes Gerät steuern kann. Neben Smartphones und Tablets könnten auch Bewegungs- und Tiefensensoren sowie Kameras zum Einsatz kommen. Dabei soll auf der Steuerungsseite keine Art der Datenvisualisierung geben.

### 2.2.1. Android Smartphone

Das mobile Betriebssystem Android läuft auf einer sehr hohen Anzahl an Smartphones weltweit. Somit können viele potenzielle User erreicht werden. Die nötigen Entwicklertools für Android stellt Google für die verschiedensten Desktopplattformen kostenlos zur Verfügung. Da auch eine gute Kenntnis über die Androidprogrammierung im Team ist, ist Android die Plattform unserer Wahl.

### 2.2.2. iOS Smartphone

Smartphones mit dem iOS Betriebssystem sind ebenfalls weit verbreitet. In der Programmierung auf dieser Plattform haben wir aber keine Erfahrungen. Außerdem müssten uns Mac-Computer zur Verfügung stehen, da Apple die nötigen Entwicklertools nur für sein Betriebssystem bereitstellt. Deshalb ist eine iOS-Version der Steuerung vorerst kein zweitrangig.

### 2.2.3. Einsatz von Bewegungssensoren und Kameras

Im Bereich der Sensoren ist der Hersteller Primesense nennenswert. Dieser stellt zu den eigenen Sensoren Treiber und Entwicklertools kostenfrei zur Verfügung. Der Preis für einen Sensor befindet sich aber im gehobenen Preissegment. Außerdem kann der Sensor nicht im lokalen Handel gekauft werden. Der Name Primesense ist auch nicht vielen Personen ein Begriff, obwohl Primesense die Sensoren für Microsoft Kinect entwickelt. Angesichts dieser Nachteile wollen wir vorerst von diesen Sensoren Abstand halten, da wir denken, dass die Applikation sonst möglicherweise kaum Verbreitung finden kann. Über eine abstrahierte Schnittstelle soll eine spätere Erweiterung für Sensoren aber möglich sein.

# 3. Paperprototypes und Interaktionen

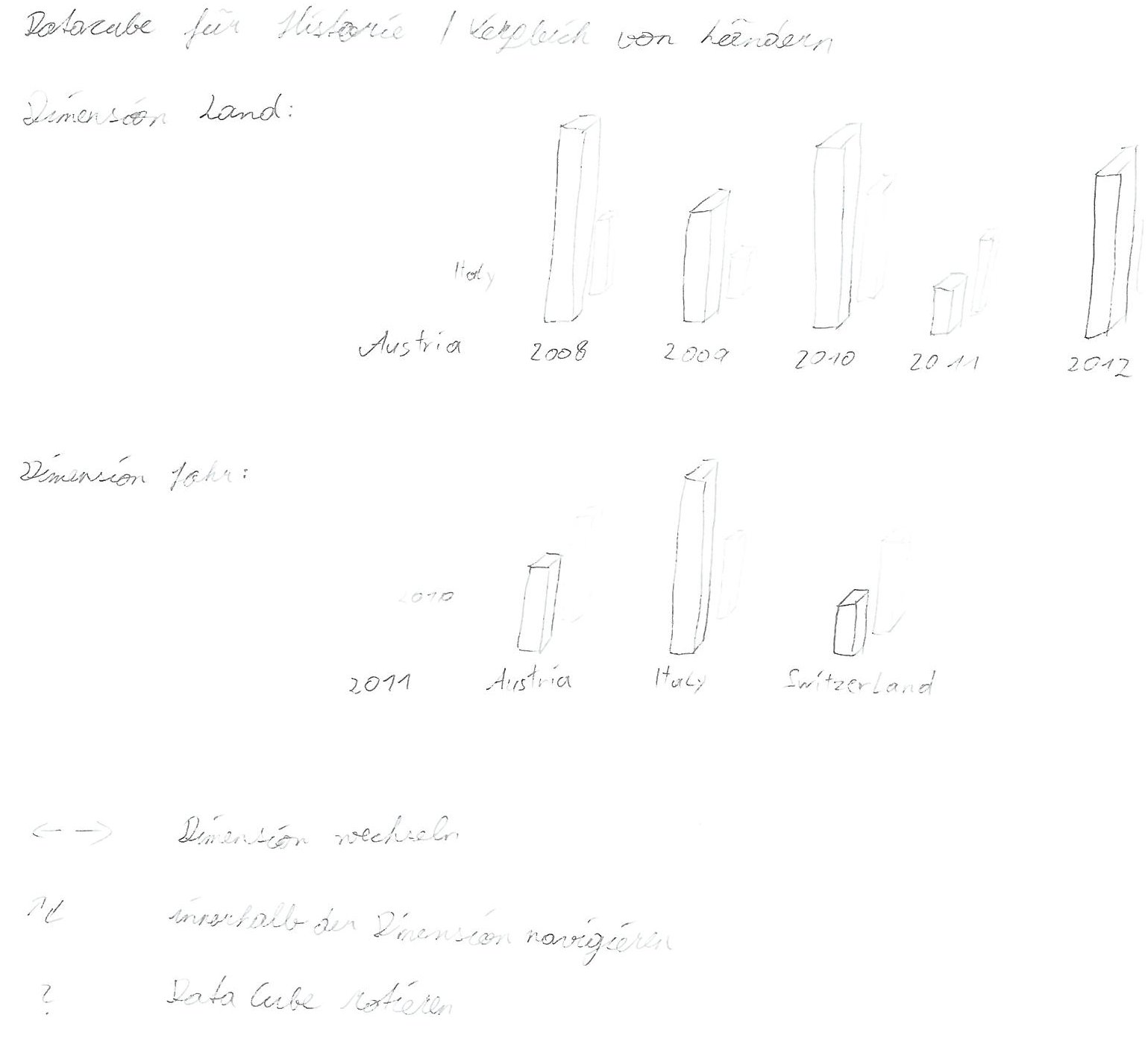
## 3.1. Erste Skizzen

### 3.1.1. Auswahl von Kontinenten, Ländern und Indikatoren



Die Auswahl von Ländern über die Kontinente und dann die Indikatoren der Weltbank enthält zu viele Ebenen. Das Auswahlverfahren basiert auf Listen. Dadurch muss der User möglicherweise sehr lange Listen durchforsten, bis er oder sie die gewünschten Einträge erreicht. Zusätzlich kommen hier keinerlei 3D-Fähigkeiten zum Einsatz.

### 3.1.2. Vergleich von Ländern

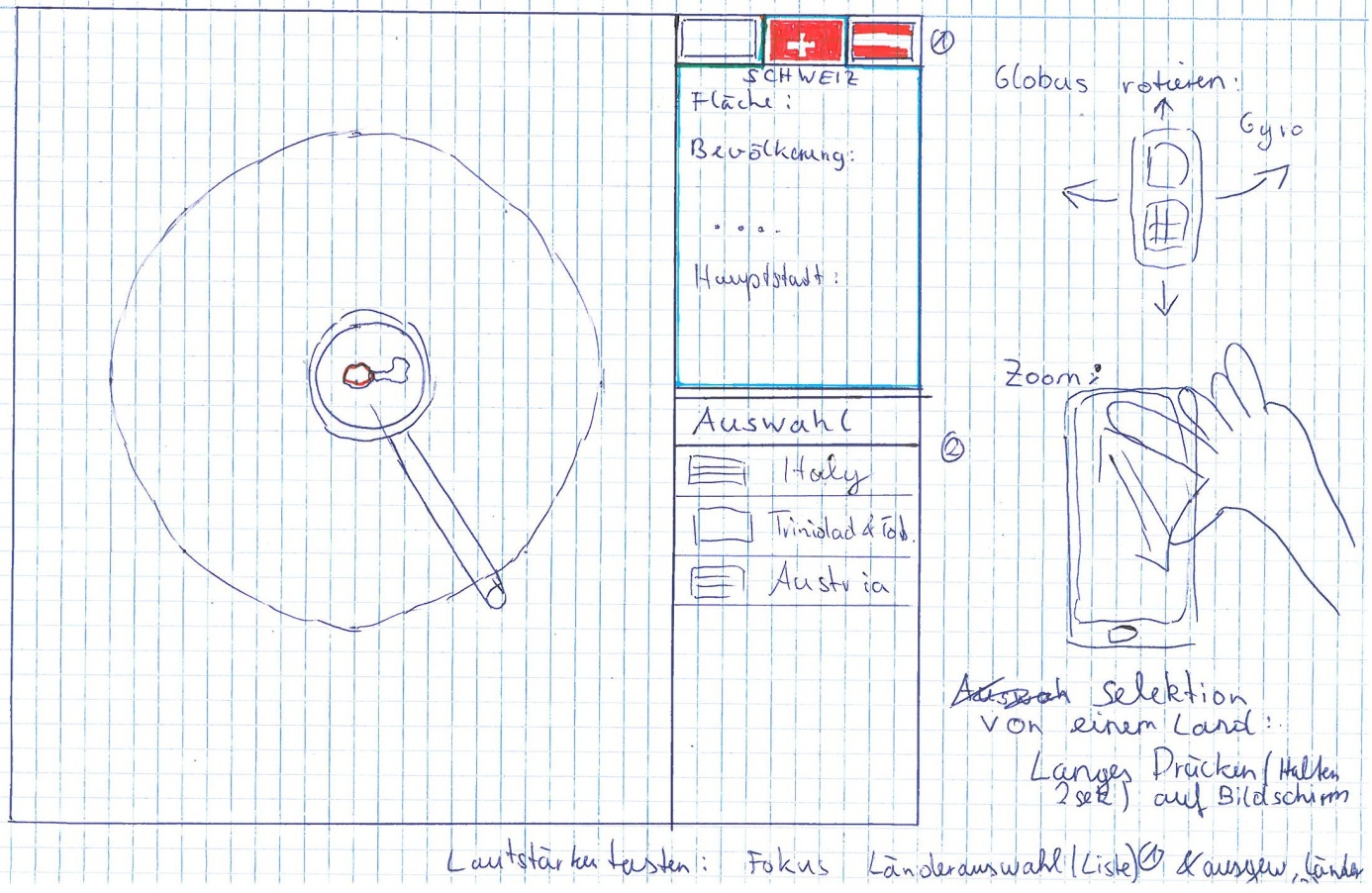


- viele Daten können irritieren

- je nach gewünschter Vergleichsart unpraktisch

## 3.2. Skizzen zum User Interface

## 3.2.1. Länderauswahl

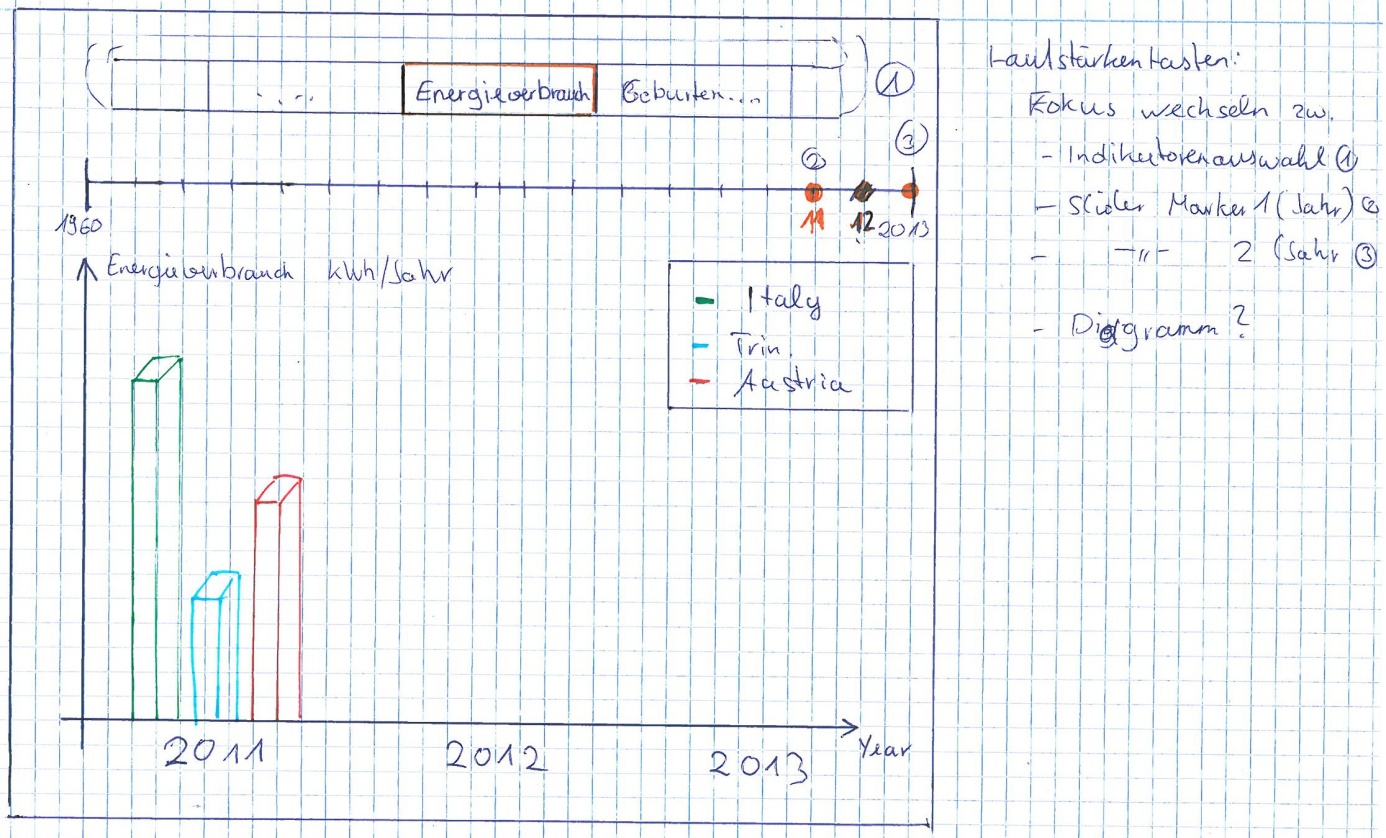


In der zweiten Version der Länderauswahl steht der Globus im Mittelpunkt. Ein Land soll mittels Rotation des Globus und Zooming gefunden und ausgewählt werden. Der Listenansatz steht alternativ für dem User unbekannte Länder bereit. Dieser bietet sich aber auch für zur Auswahl von kleinen Ländern an, die nicht gut über die Rotation zentriert werden können.

Ein Infofeld rechts oben soll grundlegende Informationen zum aktuell fokussierten Land anzeigen. Darunter gehören zum Beispiel die Hauptstadt und die Bevölkerungsanzahl.

Unten auf dem Bildschirm wird zusätzlich eine Liste der ausgewählten Länder für den Datenvergleich verwaltet.

### 3.2.2. Vergleich von Ländern



## 3.3. Interaktionen

### 3.3.1. Länderauswahl

#### 3.3.1.1. Zoom

|  |  |
| --- | --- |
| pp005.jpg | pp005.jpg |
| bekannte **Zoomgeste** von Smartphones (zwei Finger spreizen oder zusammenziehen) | |

#### 3.3.1.2. Fokus wechseln

|  |  |
| --- | --- |
| pp005.jpg | pp005.jpg |
| Durch nach **Oben/Unten Wischen** am Smartphone (von manchen Apps auch zum Scrollen verwendet) | |

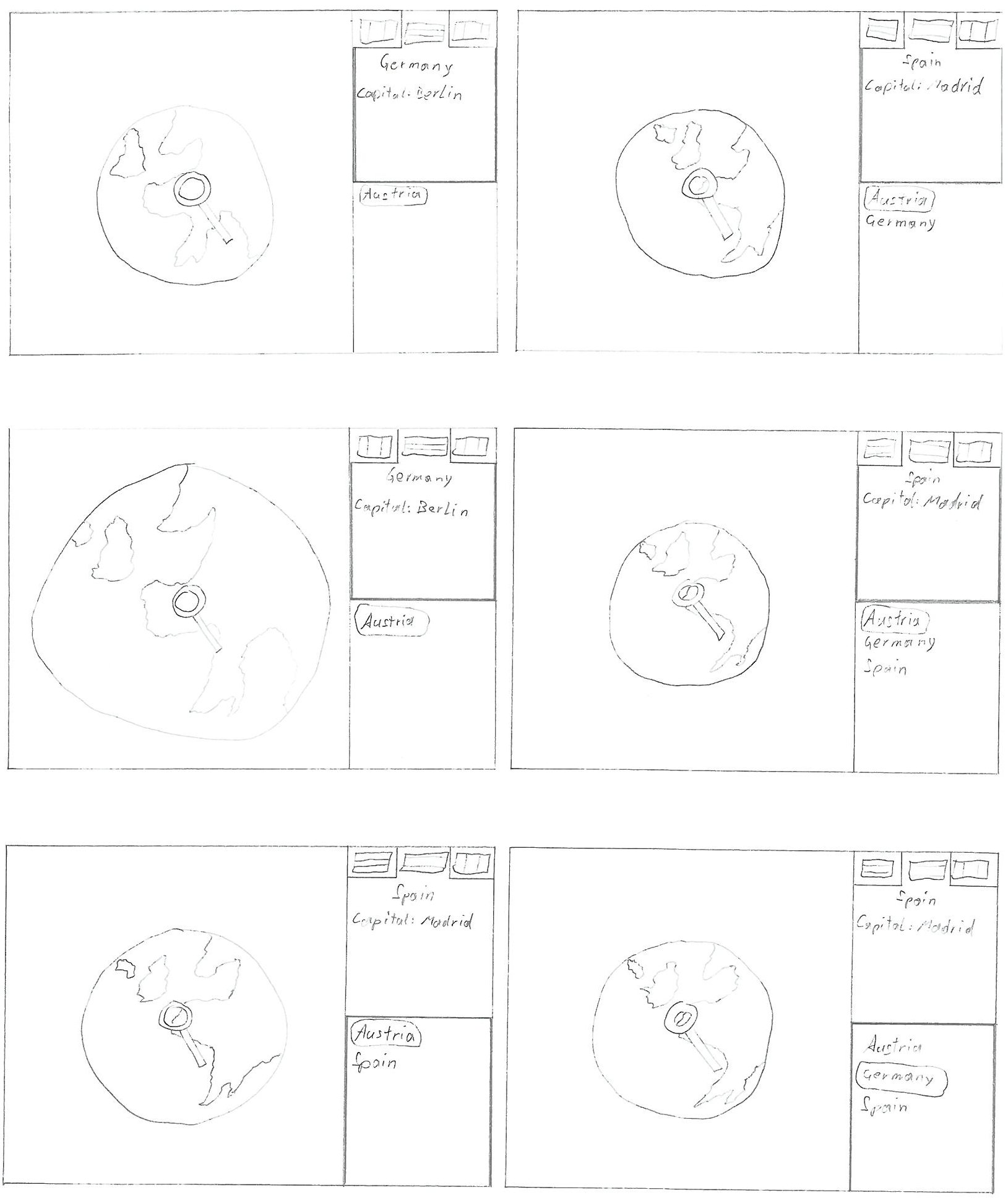
#### 3.3.1.3. Land auswählen

|  |  |
| --- | --- |
| pp005.jpg | pp005.jpg |
| Smartphone rotieren um Globus zu drehen  oder  Fokus auf Länderauswahl oben gesetzt und **horizontal wischen** | |

#### 3.3.1.4. Land zur Vergleichsauswahl hinzufügen

|  |  |
| --- | --- |
| pp005.jpg | pp005.jpg |
| Fokus auf Länderauswahl oben  bei Auswahl des gewünschten Landes  **Doppelklick** auf den Bildschirm vom Smartphone (typische Auswahlaktion bei PCs) | |

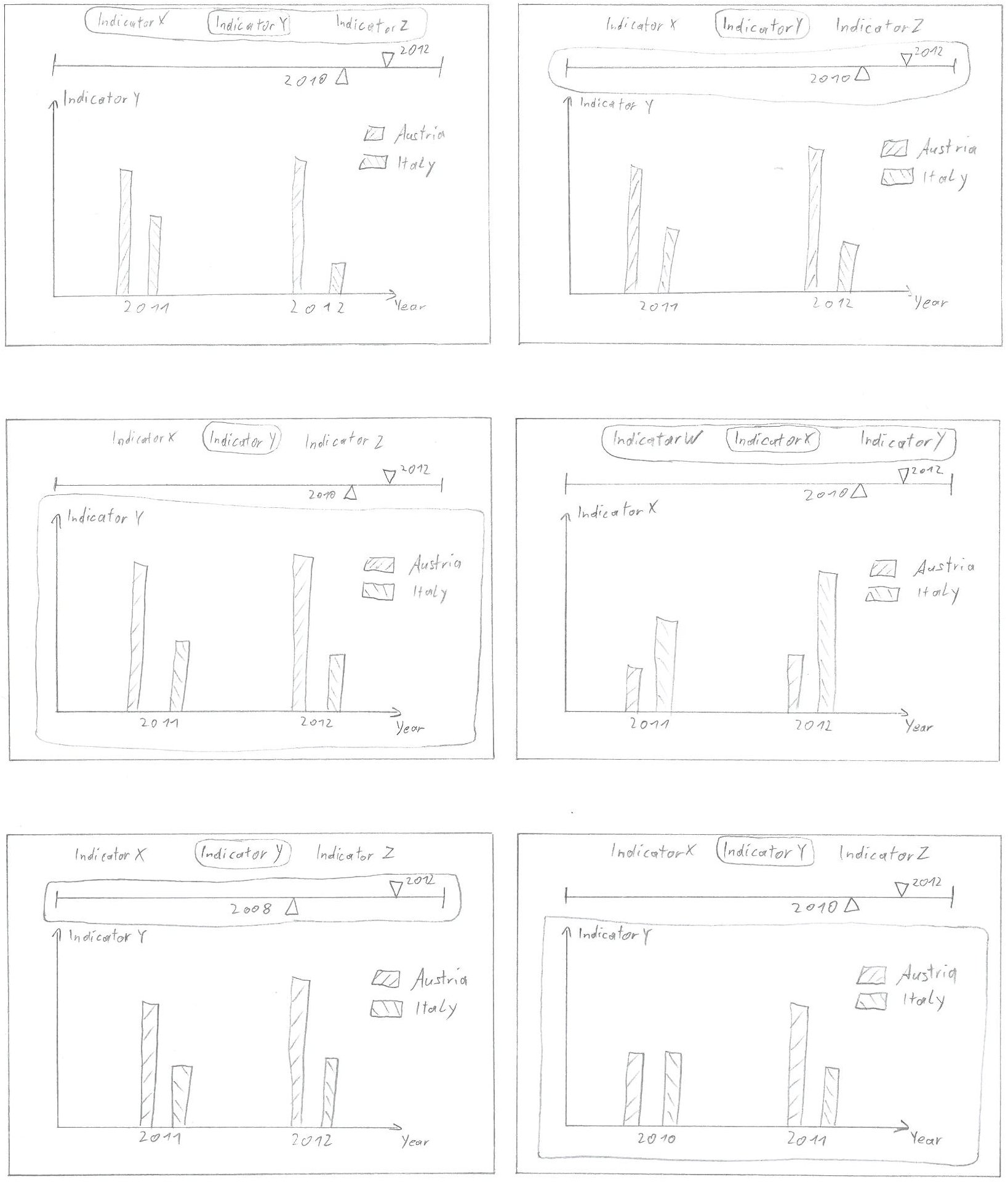
#### 3.3.1.6. Land aus der Vergleichsauswahl entfernen



Mittels **Swipe nach Oben / Unten** wird das Land aus der Auswahl selektiert.

Mit einem **Doppeltouch** wird das Selektierte entfernt.

### 3.3.2. Datenvisualisierung



Beinhaltet:

* Indikatorenauswahl
* Jahreseinschränkung
* Diagramm

# 4. Realisierung

## 4.1 Verwendete Tools

* Unity3D (v 4.3)
* Google Android SDK (SDK 15)
* Visual Studio
* Eclipse
* Blender

## 4.2 Unity-Anwendung

Die Unity-Anwendung dient hauptsächlich zur Darstellung der Weltkugel und Visualisierung von Daten.

Die Anwendung wurde komplett auf einem Domänenmodell aufgebaut. Dadurch ist die Darstellung, Datenverarbeitung, Kommunikation im Code voneinander getrennt. Dies führt zur besseren Erweiterbarkeit oder Änderbarkeit der Anwendung.

### 4.2.1 Ländererkennung und Koordinatenumrechnung

#### Ländererkennung

Das aktuell dargestellte Land ist bei Globetrotter immer das Land auf das die Kamera zeigt. (Kamera ist auf Mittelpunkt vom Globus zentriert.

Um eine Position auf der Sphere zu bekommen haben wir den Raycast von Unity verwendet. Damit eine Kollision zustande kommen kann muss die „Sphere“ im Unity unbedingt einen **MeshRenderer** besitzen.

|  |
| --- |
| // Mitte von Kamera als Ausgangsposition  Ray ray = Camera.main.ViewportPointToRay (new Vector3(0.5f,0.5f,0));  RaycastHit hit;  // Nach Kollision mit Sphere schauen  if (collider.Raycast (ray, out hit, 10000f)) {  // Kollision mit Sphere (MeshRenderer)  // Umrechnung der Koordinaten auf Originalgröße der Textur  int w = (int)Mathf.Round(hit.textureCoord.x \* 4096);  int h = (int)Mathf.Round(2048- hit.textureCoord.y \* 2048);    Vector2 v = PixelXYToLatLong(w, h);  Country c = getCountry(v.y, v.x);  if(c != null){  …  }  } |

#### Rotieren zu einem Land

Prinzipiell werden die GPS-Koordination mit denselben Formeln umgewandelt wie bei der Erkennung von Ländern. Damit jedoch die Kugel an die gewünschte Position rotiert werden kann muss man die Texturkoordinaten in den 3D Raum von Unity umrechnen.

|  |
| --- |
| private void rotate(){  Country c = null;  \_countries.TryGetValue(m\_countrySelectorViewModel.CurrentCountry.IsoAlphaThreeCode, out c);    if(c != null)  {  Debug.Log(c.Longitude + " - " + c.Latitude);  // Umrechnung von Latitude / Longitude auf x,y  Vector2 v2 = LatLongToPixelXY(c.Latitude, c.Longitude);  // Umrechnung der Koordinaten für Unity  v2.x = v2.x/ 4096;  v2.y = (v2.y - 2048) / 2048 \* - 1;  // Umwandlung der 2D Koordination in den 3D Raum  Vector3 vector = UvTo3D(v2);  Debug.Log(vector.x+"-"+vector.y+"-"+vector.z);    lock(m\_lockObj)  {  // Vektor der die Rotation der Kugel angibt  m\_rotationVector = vector;  m\_rotateToCountry = true;  }  }  } |

#### UvTo3D

Diese Funktion wird verwendet um die 2D Tekturkoordinaten in die Koordinaten des Spherengitters umzuwandeln.

|  |
| --- |
| Vector3 UvTo3D(Vector2 uv) {  Vector2[] uvs = m\_uv;  Vector3[] verts = m\_vertices;  int[] tris = m\_triangles;  for (int i = 0; i < tris.Length; i += 3){  Vector2 u1= uvs[tris[i]];  Vector2 u2= uvs[tris[i+1]];  Vector2 u3= uvs[tris[i+2]];    float a = Area(u1, u2, u3); if (a == 0) continue;      float a1= Area(u2, u3, uv)/a; if (a1 < 0) continue;  float a2 = Area(u3, u1, uv)/a; if (a2 < 0) continue;  float a3 = Area(u1, u2, uv)/a; if (a3 < 0) continue;    Vector3 p3D = a1\*verts[tris[i]]+a2\*verts[tris[i+1]]+a3\*verts[tris[i+2]];    return p3D;  }    return Vector3.zero;  }  float Area(Vector2 p1, Vector2 p2, Vector2 p3){  Vector2 v1= p1 - p3;  Vector2 v2 = p2 - p3;  return (v1.x \* v2.y - v1.y \* v2.x)/2;  } |

#### PixelXYToLatLong und LatLongToPixelXY

Diese Funktion rechnet die Texturkoordinaten in Latitude, Longitude um. Sie berechnet die GPS-Koordinaten zwar nicht ganz genau, jedoch sind diese Fehler nur bei den Polarkappen groß. Ansonsten ist es eine gute Annäherung zur aufwendigeren Berechnung.

|  |
| --- |
| Vector2 PixelXYToLatLong(int pixelX, int pixelY)  {  Vector2 vector = new Vector2();  float longitude = pixelX / (4096/360.0f) - 180;  float latitude = (pixelY / (2048/180.0f) - 90) \* -1;  vector.x = latitude;  vector.y = longitude;  return vector;  } |

|  |
| --- |
| Vector2 LatLongToPixelXY(float lat, float lon){  Vector2 vector = new Vector2();  vector.x = (4096/360.0f) \* (180 + lon);  vector.y = (2048/180.0f) \* (90 - lat);  return vector;  } |

#### GetCountry

Diese Funktion ermittelt ein Land anhand von Longitude und Latitude.

Die Grenzen aller Länder der Welt sind in einer CSV-Datei hinterlegt, die beim Start ausgelesen wird.

Der Algorithmus untersucht ob Werte in einem Polygon liegen oder nicht. Bei Globetrotter wird jedes Land daher als ein einzelnes Polygon oder als mehrere Polygone definiert.

Der dazugehörige Algorithmus ist hier erklärt:

<http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html>

#### 4.2.2 Anbindung zum Weltbank-Webservice

Für die Verwendung des Services gibt es eine ausführliche Dokumentation der Weltbank:

<http://data.worldbank.org/developers/api-overview>

Für Klimadaten:

<http://data.worldbank.org/developers/climate-data-api>

#### 4.2.3 Chart-Service

Für die Generierung der Diagramme läuft im Hintergrund der Applikation ein Java-Webservice. Dem Service müssen die gewollten Länder, Jahre, Indikatoren übergeben werden. Nach dem Abrufen der Weltbankdaten generiert dieser ein Diagramm, das als Bild zurückgegeben wird.

### 4.3 Android-App

Die Globetrotter Applikation liest Sensorwerte aus und erkennt Gesten. Diese Werte bzw. Informationen zu Gesten werden über eine UDP-Datenübertragung an die Unity-Anwendung gesendet.

Zur Initialisierung der Verbindung muss der Benutzer einfach den QR-Code mit der App scannen.

# 5. Probleme während der Entwicklung und deren Lösung

## 5.1. Anfrage an den Webservice der Weltbank

Am Anfang hat die Datenabfrage an den Webservice der Weltbank ganz gut funktioniert. Aber über Weihnachten stellten wir fest, dass die Weltbank scheinbar einige Änderungen am Webservice vornahm.

Es ist uns fiel auf, dass der Webservice nach der Datenabfrage die Connection bei HTTP/1.1 offen hält. Daraus resultierte, dass unser Code das Ende der Datenübertragung nicht mehr erkennen konnte und weiter auf Daten wartete. Durch die Berücksichtigung der *Content-Length* im Header konnten wird aber die genaue Datenmenge herausfinden und nur solange Daten einlesen, bis wir diese Menge erreichten.

Den HTTP-Request mussten wir ebenfalls umstellen. Der Webservice zur Abfrage von Indikatoren schickte bei unserer alten Implementierung auf einmal den Fehler *400 Bad Request*. Der Webservice zur Abfrage der Länder nahm den HTTP-Request aber als korrekt an. Dieses Problem lässt sich aber beheben, indem jede Zeile im HTTP-Request konsequent mit den *\r\n* beendet wird. So werden die HTTP-Request von den Webservices richtig angenommen.

## 5.2. Zufälligerweise 502 Bad Gateway

Nun tritt manchmal und für uns zufälligerweise ein neuer Fehler auf. Manchmal passiert es, dass der Webservice den Fehler *502 Bad Gateway* zurückschickt. Bei einer erneuten Sendung desselben HTTP-Requests wird dieser vom Webservice wieder al korrekt angenommen. Dieses merkwürdige Verhalten vom Webservice können wir aber mit einer Prüfung auf diesen Fehler und einer erneuten Übertragung des HTTP-Request handhaben.

## 5.3. Textencoding in Unity

- bei XML

- ohne BOM (byte order mark) am Anfang sonst merkwürdige Exception

- System.Xml.XmlException: Text node cannot appear in this state. Line 1, position 1.

- mit MemoryStream read und danach write gelöst

- Fehler in C# Doku (<http://msdn.microsoft.com/en-us/library/system.io.bufferedstream%28v=vs.110%29.aspx>): falsche Offsets

TODO

## 5.4. Smartphone, Datenübertragung

Testgeräte:

* Samsung Galaxy S2
* HTC One

Sensoren unterschiedlicher Geräte liefern unterschiedlich viele Werte, beispielsweise sendet der Lagesensor des „Samsung Galaxy S2“ ungefähr 60 mal soviel Werte in der Minute wie ein „HTC One“.

Das unterschiedliche Verhalten zeigt sich auch bei der Gestenerkennung. Beispielsweise werden bei der Zoomgeste ständig Informationen gesendet um damit den Faktor des Zooms festzustellen.

Als Folge der unterschiedlichen Anzahl an Werten gab es bei der Performance (ruckelfreie Darstellung, Stabilität, etc.) der Unity-Anwendung je nach Smartphonetyp große Unterschiede.

Um die Anzahl der Werte zu beschränken bzw. zu normalisieren wird die Anzahl von der Android Applikation beschränkt. (20 Werte / Sekunde)

Durch die Beschränkung der Werte wurde auch die Performance der Datenübertragung stark verbessert.

# 5. Systemanforderungen

|  |  |
| --- | --- |
| **Hardware** | |
| Smartphone | Android |
| Netzwerk |  |
| **Globetrotter** | |
| Betriebssystem | Windows oder Mac |
| Bildschirmauflösung | Empfohlen: 1280 x 720  16:9 Seitenverhältnis für korrekte Darstellung erforderlich |
| **Globetrotter-Steuerungsapp** |  |
| Betriebssystem | Android SDK 15 (Android 4.0.3) |
| Sensoren | Accelerometer |