Südtirol Graffiti

Von Titus Ramyzkowski und Robert Böing

Inhaltsverzeichnis

1. Suatuland Graffiti

Suatuland Graffiti soll das Smartphone zur Sprühdose machen. Virtuelle Graffitis sollen an Häuserwänden, U-Bahnschächten und an Zügen entstehen, um diese auf seinem Handy Freunden und Bekannten zu zeigen.

1.1 Ideenfindung

Eine Graffiti-App nutzt alle Sensoren eines Smartphones in hohen Maßen. Daher war die reine Möglichkeit der Grund des Projektes.

Eine Spraydose im echten Leben braucht einiges an Informationen um ein schönes Graffiti an eine Wand zu *bomben*[1]. Da Spraydosen cool sind und der Nerdfaktor der Informatik immer noch groß ist, war die Idee von „Suatuland Graffiti“ festgeschrieben.

Als Idee sollte auch die Community nicht fehlen. Eine Virtual Reality Anwendung macht die mit Hilfe der Kamera und einem Marker bereits gemalten Graffitis sichtbar. Dies sollte anonym geschehen. Die GPS-Koordinaten eines Graffitis würden gespeichert und könnten in Kombination mit einem Markierungspunkt alle nötigen Informationen zur Verfügung stellen, um die Zeichnung korrekt auf einem fremden Smartphone anzuzeigen. Speichern sollte man diese Informationen nach einem ähnlichen Prinzip, lediglich die GPS-Koordinaten des Smartphones würden an den Suatuland Server geschickt.

1.2 Konzeption

In der ersten Phase der Konzeption wurde die technische Machbarkeit des Projektes überprüft. Die Recherchen ergaben, dass zur Auswertung einer „Sprühdose“ die Beschleunigung und die Lage des Smartphones notwendig sind.

Siehe Sensorik

Die Community-Features wurden als *nice to have* deklariert. Dies war notwendig, da schnell klar wurde, dass sich die Umsetzung der *must haves* als sehr umfangreich erweisen würde.

Ein großer Teil der Arbeit sollte von Anfang an ins Layout fließen. Es sollte eine App entstehen, die sich von bekannten Formen abhebt und quasi das Image einer Sprühdose darstellt.

[1] Graffiti an die Wand sprühen

2. User Interface und Bedienung

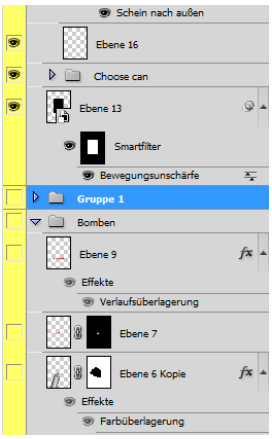

Abbildung I

Für das Interface mussten sämtliche Views, wie in Abbildung I zu sehen, als Mockups[2] gebaut werden. Die Umsetzung stellte sich mangels flexibler Auszeichnungssprache als recht umständlich heraus.

So kam es zu einem Mix an eingesetzten Techniken. Einfache Views wurden mit der in Android spezifizierten XML Darstellung realisiert, andere, kompliziertere Views wurden als Bitmap gezeichnet.

2.1 Beispiele

Alle Beispiele sind am Ende der Dokumentation nochmal in hoher Auflösung aufgeführt.

Abbildung II

2.1.1 Willkommensbildschirm (Abb. II)

Der Erste View ist das Menü, bei dem der Nutzer entscheidet, ob er in die Galerie zum *Glotzen,* oder in den Sprühmodus zum *Bomben* gelangen möchte.

Dieser View wurde als XML-Markup realisiert. Man sieht ein großes Hintergrundbild, das Auto mit Wand und Herz. Zusätzlich sind zwei Buttons, *Glotzen* und *Bomben* dargestellt.

[2] Entwurf zur Darstellung einer Benutzeroberfläche

Abbildung III

2.1.2 Spraymenü (Abb. III)

In diesem Menü ist es möglich, ein gerade gesprühtes Bild zu speichern oder eine andere Farbe auszuwählen.

Auch dieser View wurde als XML-Markup realisiert. Damit diese Ansicht eindeutig dem Sprayview zugeordnet werden kann, wurde das Menü mit einem halbtransparenten Hintergrund versehen.

Hinter beiden Menüpunkten befinden sich keine neuen Actions oder Views, lediglich Dialoge, die mit entsprechender Rückgabelogik ihre Funktionen über das Hauptprogramm ausführen.

Abbildung IV

2.1.3 Sprayview (Abb. IV)

Die Darstellung während des Sprühens war oberflächentechnisch die anspruchsvollste Aufgabe. Sie wird auf der einen Seite komplett als Bitmap gezeichnet und enthält sich ständig ändernde Elemente. Nach der Auswahl einer Farbe werden die Sprühdose und der daraus resultierende Farbkleks neu gezeichnet.
Im Hintergrund ist das Kamerabild zu sehen. Um diesem ein Graffiti-Look zu verpassen, ist eine mit Alphatransparenz abgestimmte Mauer über das Bild geblendet.

3. Ressourcen

Um das Gefühl einer Spraydose zu unterstützen und die Funktion der App sicherzustellen, werden Unterschiedliche Ressourcen benötigt.

- **Kamera**
  Die Kamera wird benötigt, um den realen Hintergrund auf dem Display des Handys darzustellen. Dieses bewegte Bild dient als Hintergrund des Graffitis.

- **Vibrator**
  Um die Mischkugel einer echten Sprühdose zu simulieren, gibt es nach jedem Schütteln des Handys, ein kurzes Vibrationsfeedback.

- **Sensoren**
  Als Eingabemethode dienen Bewegungs- und Richtungssensoren. Siehe Sensorik

4. Entwicklungsumgebung

Das Projekt wurde mit Hilfe des Android Software Developer Kits in der Version 2.1-update1, API 7, revision 2 umgesetzt. Hierbei kam Eclipse als IDE zum Einsatz. Weiterhin wurde die Applikation direkt auf den Endgeräten getestet, da auf Grund der Verwendung des Bewegungssensors als „Zeichengerät“, der Emulator keine bzw. keine ausreichenden Ergebnisse liefern würde.

5. Sensorik

5.1 Wozu ein Sensor?

Für die Realisierung des Projekts *Suatuland Graffiti* sind die Sensordaten des verwendeten Endgerätes essentiell. Der Grundgedanke hierbei war die Verwendung eines Bewegungssensors, der in jedem mit Android ausgestatteten Mobiltelefon, zur Berechnung der aktuellen Position des Gerätes bzw. des zurückgelegten Weges, verbaut sein muss.

5.2 Implementierung in Android

Ein Bewegungs- bzw. Beschleunigungssensor misst die Beschleunigung in festgelegte Richtungen. In unserem Fall geht es um die Richtungen eines dreidimensionalen kartesischen Koordinatensystems: x, y und z – hierbei handelt es sich um die am meisten verwendete Implementierung. Dementsprechend bekommen wir bei einer Anfrage an den Sensor im Telefon ein Array zurückgeliefert, welches uns mit der Beschleunigung in die jeweiligen Richtungen versorgt. Bei values[0] handelt es sich um die Beschleunigung in x-Richtung, values[1] versorgt uns mit dem Wert für die y-Richtung und bei values[2] handelt es sich um die Beschleunigung in z-Richtung.

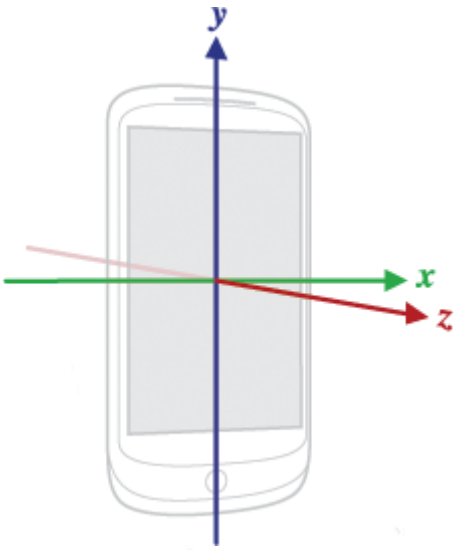

Abbildung V

5.3 Umrechnung ins Welt-System

Die zurückgelieferten Werte beziehen sich auf das System des Mobiltelefons. Dies bedeutet, dass die y-Achse das Gerät der Länge nach teilt, während die x-Achse orthogonal dazu, das Gerät in der Breite durchschneidet. Um die Bewegung des Telefons nachzuvollziehen, müssen wir die vom Bewegungssensor zurückgelieferten Werte in unser Koordinatensystem (das „Welt-System“) übertragen. Dadurch soll eine Verfälschung der berechneten Positionsdaten durch Drehung des Geräts um die eigene Achse vorgebeugt werden. Dies kann mittels Rotationsmatrizen und der Achsenbeschleunigung realisiert werden. Die Drehung des Geräts um die eigene Achse brauchen wir allerdings nicht nur, um Fehler bei der Positionsbestimmung nachvollziehen zu können, sondern auch, um die Erdanziehungskraft aus den vom Beschleunigungssensor zurückgelieferten Daten zu entfernen. Da sich momentan noch kein Gerät mit einem Gyroskop auf dem deutschen Markt befindet, muss die Achsenbeschleunigung (Die Beschleunigung um die einzelnen Achsen) mit Hilfe des Beschleunigungssensors und des im Gerät verbauten Kompasses berechnet werden.

6. Projektverlauf

6.1 Projektfindung

Die Entscheidung für *Suatuland Graffiti* kam relativ spontan. Wir überlegten uns schon seit Wochen welche Art von Projekt wir umsetzen wollten. Eines war dabei immer klar: Es sollte keine mobile Anwendung werden!

6.2 Der Beginn

Nachdem die Entscheidung für die Applikation gefallen war, machten wir uns an die Umsetzung. Wie rechnet man Beschleunigungsdaten in Positionsdaten um? Welche Schwierigkeiten können dabei auftreten? Welchen Funktionsumfang soll die Anwendung einmal besitzen?

Die theoretische Anwendung war relativ leicht. Aus dem Physikunterricht weiß man: Beschleunigung = Geschwindigkeit * Zeit. Da uns sowohl die Beschleunigung als auch die Zeit bekannt ist, konnten wir durch das Umstellen dieser Basisformel die Geschwindigkeiten für die einzelnen kleinen Zeitintervalle errechnen. Diese wurden in die zurückgelegten Strecken umgerechnet und durch Addition der einzelnen Streckenintervalle konnte der Weg des Gerätes verfolgt werden.

6.3 Erste Programmierarbeiten

Die Programmierung selbst begannen wir mit einigen Spielereien, um in das System hineinzufinden. So war uns von vorne herein klar, dass eines der wichtigsten Features der Sound beim Schütteln des Geräts sein würde – da dies allein schon sehr großen Spaß macht. Die Implementierung war erstaunlich einfach:

```
int direction = getDirection(values[1],y);

if(length > 6 && direction == 1) {
      if(played != 1) {
            base.mSoundManager.playSound(3, 0, 1);
            played = 1;
            base.mDildroid.vibrate((long) 30);
      }
}

private int getDirection(float last, float cur) {

      if(last - cur < 1.5) {
            return -1;
      } else {
            if (last - cur > 1.5) {
                  return = 1;
            } else {
                  return = 0;
            }
      }
}
```

Gleichzeitig begann die Arbeit am User Interface. So wurden erste Entwürfe der Menüs, sowie der Interfaces in der Anwendung selbst erstellt.

6.4 Die Sache mit der Erdanziehung

Doch bereits beim nächsten Programmierschritt traten Probleme auf. Wir schrieben die Sensordaten in eine Datei auf der SD-Karte und haben mit der Auswertung der Daten

begonnen. Für die visuelle Darstellung wurde schnell eine Anwendung mit grafischer Oberfläche geschrieben. Das erste Ergebnis war überraschend:

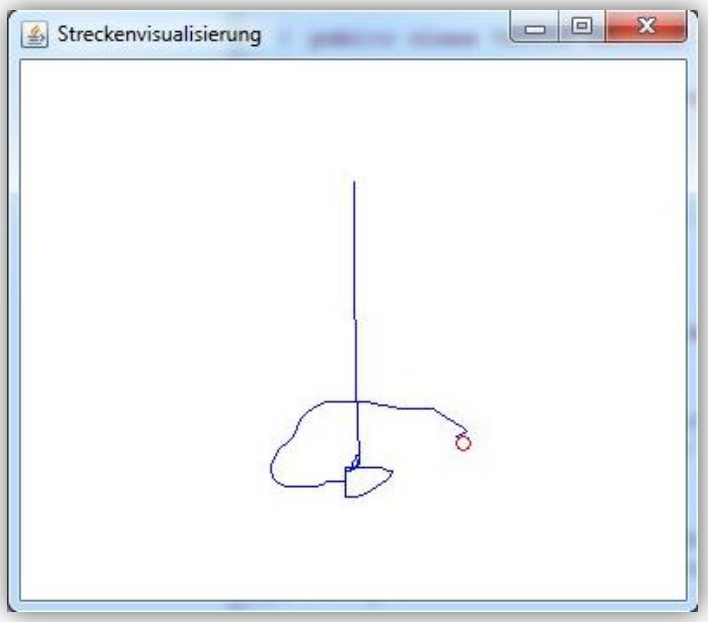

Abbildung VI: Visualisierung einer Kreisbewegung

Das Problem war der Highpass-Filter zur Eliminierung der Gravitation, wie er in der Android Dokumentation empfohlen wurde. Dieser benötigte über eine Sekunde, um die momentane Richtung der Gravitation zu berechnen und sie von den einzelnen Werten abzuziehen. Ebenso sind bei dieser Implementierung geringere Beschleunigungen bzw. kürzere Strecken z.B. für Feinarbeiten nicht möglich, da sie durch die benötigte Länge und Intensität einer Bewegung für die Änderung der Werte des Highpass-Filters einfach ignoriert wurden. Er war für unsere Zwecke also schlicht unbrauchbar.

Was also tun? Nach langem Studium verschiedener Dokumentation, Internetforen und der Fachliteratur kamen wir zu der Erkenntnis, dass die Eliminierung der Gravitation mit nur einem Sensor schlichtweg unmöglich ist. Immer wieder stießen wir auf dieselbe Antwort: „Es ist nicht möglich aus den wirkenden Kräften herauszufinden, aus welchem Grund eine Kraft wirkt. Einstein hat dies in der Relativitätstheorie bewiesen!“[3]

[3] Nach: http://stackoverflow.com/questions/3377288/how-to-remove-gravity-factor-from-accelerometer-readings-in-android-3-axis-accele

6.5 Versuch der Orientierung

Gut, wenn es nicht mit nur Einem Sensor funktioniert nehmen wir eben Zwei: Wie bereits erwähnt, lässt sich zwar die Orientierung des Geräts mit dem Beschleunigungssensors unter Zuhilfenahme des Kompasses berechnen. Die zurückgelieferten Werte allerdings sind nicht zu gebrauchen, da bei der Bewegung des Geräts selbst, die errechnete Drehung um die eigene Achse stark verfälscht wird. Und schließlich basiert die Eingabe unserer Anwendung auf der Bewegung!

Diverse andere Ansätze, wie z.B der Einsatz der Kamera zur Bewegungserkennung bzw. dessen Unterstützung, wurden auf Grund von großen Einschränkungen oder da sie schlichtweg nicht realisierbar waren, schnell wieder verworfen.

6.6 Scheitern des Hauptziels

An diesem Punkt wurde uns klar, dass das Projekt – wie wir es uns vorstellten – nicht zu realisieren war. Wir entschieden uns also dazu, die Gravitationsproblematik durch die Verlegung des Systems in eine Ebene (Sprich: die Tischplatte) zu umgehen und dem User Interface mehr Aufmerksamkeit zu schenken, da sich in Zukunft eine Lösung des Problems ergeben könnte.

6.7 Die Alternative

Auch in der Ebene ergaben sich aber weiterhin Probleme. So kam es oft genug dazu, dass das Gerät sich zwar nicht mehr bewegen lies, die errechnete Geschwindigkeit aber auf Grund des Rauschens der Sensordaten oder durch versehentliche minimale Drehungen des Geräts um die eigene Achse, nicht mehr auf Nullgesetzt wurde und dadurch Bewegungen ins Unendliche entstanden. Dies lösten wir durch Versuche, ab welchem Ausschlag eine tatsächliche Bewegung -und kein Rauschen- vorlag. Dazu konnten wir unter Zuhilfenahme der Länge des Beschleunigungsvektors minus der Gravitationskraft feststellen, ob sich das Gerät bewegte oder regungslos auf dem Tisch lag. Allerdings konnten diese Maßnahmen, die durch Messfehler und Rauschen verursachte Problematik nicht lösen. So mussten wir ein Auslaufen der Werte durch eine Mindestbewegungsschwelle und der konstanten Verringerung der Geschwindigkeit verhindern.

6.8 Das Interface und die APIs

Da die ursprüngliche Anwendung im dreidimensionalen Raum geplant war, ließen wir die Kamera aktiv (was bei der Anwendung auf dem Tisch natürlich nicht auffällt) und zeichneten unser Interface auf diesen Kamerahintergrund. Hierbei stießen wir auf das nächste Problem: Unsere Applikation ist für die senkrechte Haltung des Geräts designt. Bis API 8 (Android 2.2) befindet sich allerdings ein Bug im System, welcher die horizontale Anwendung der Kamera verhindert und das Bild verzerrt ausgibt. Deshalb mussten wir unser komplettes Interface und die darauf gezeichneten Koordinaten um 90° drehen, um später ein vertikales Interface „vorzugaukeln“. Die Verwendung API 8 kam nicht in Frage, da für eines unserer Testgeräte nur API 7 (Android 2.1-update1) verfügbar war.

Durch die entstandenen Probleme und den resultierenden Zeitaufwand, konnten wir Suatuland Graffiti bis zur Medianight nur in den Hauptfunktionalitäten fertigstellen.

7. Zukunftsaussichten

Mit API 9 (Android 2.3) erhalten zwei neue Sensoren Einzug in das mobile Betriebssystem. Zum einen werden nun Gyroskope unterstützt, die die Achsengeschwindigkeit, und damit die Drehung messen können, sowie Gravitationssensoren (Diese werden wohl meist durch das Gyroskop realisiert werden) unterstützt. Hierdurch werden das Entfernen der Erdanziehungskraft, sowie die Messung der Rotation möglich. Damit ergeben sich neue Möglichkeiten für die aufgetretene Problematik. Sicher ist allerdings auch, dass dies trotz allem ein äußerst schwieriges und kompliziertes Unterfangen wird. Es müssen alle, sich im Telefon befindlichen Sensoren (Beschleunigungssensor, Gyroskop, Kompass) zusammengeschaltet werden, um sich gegenseitig zu kalibrieren. Dies ist möglich, da ihre Fehlerquellen in verschiedenen Bereichen liegen – sie jedoch auch eine Schnittmenge in ihrer Funktionsweise und ihren gelieferten Ergebnissen aufweisen.

Das resultierende Ergebnis wiederum muss mittels aufwendigen Wahrscheinlichkeitsalgorithmen auf Fehler geprüft werden. Die von uns gefundenen Diplomarbeiten, beschäftigen sich mit der Entwicklung dieses Algorithmus.

Willkommensview

Abbildung VII

Abbildung VIII

Abbildung IX