Studienarbeit Mobil & Ubiquitous Computing

Sommersemester 2020

Tobias Schotter, Medieninformatik

1. Grundlegende Beschreibung der ursprünglichen AccelerometerPlay-App

Anfangs muss angemerkt werden, dass das Android-Projekt mit einer „Empty Activity“ erstellt wurde, im Gegensatz zur in der Angabe beschriebenen „Basic Activity“, da es bei der Unteraufgabe 4.a (Erstellung eines Menüs) Komplikationen gegeben hat, welche mit einer „Empty Activity“ nicht aufgetreten sind.

Zu Beginn wurden die entsprechenden Dateien aus der ursprünglichen AccelerometerPlay-App in die Empty Activity kopiert und angepasst, um diese zum Laufen zu bringen. Darunter fielen anfangs die „main.xml“ Datei, die vorerst nicht extra angepasst werden musste. Es mussten lediglich die entsprechenden Bilder in den app/res/drawable Ordner eingefügt werden. Anschließend musste das „AndroidManifest.xml“ angepasst werden, dort wurden gewisse „user-permissions“ sowie kleinere Anpassungen im „Activity“ Tag geändert.

Nun zur Hauptdatei, der „AccelerometerPlayActivity.java“. Hier wurde erstmal das „extends Activity“ zu einem „extends AppCompatActivity“ geändert, aufgrund der oben angemerkten Erstellung einer „Empty Activity“. Dies ist allerdings nur für spätere Aufgaben relevant und macht für die ursprüngliche App keinen Unterschied.

Nun lief die App schon einmal. Bloß gab es noch ein Problem, denn die „Kugeln“/“Partikel“ waren richtig eingebunden allerdings noch nicht zu sehen. Es wurde ein emuliertes Pixel 2 verwendet und die Standartgröße der Kugeln war zu klein, um diese sehen zu können. Dieses Problem zu finden hat eine Weile gedauert.

Ich habe die Entwicklung hindurch mit einer erhöhten Größe gearbeitet, bis ich am Ende herausgefunden habe, dass es an der Version meines emulierten Gerätes lag. Anscheinend brauch man unbedingt die API-Version 28. Ich fand allerdings, dass die Geschwindigkeit mit den größeren Kugeln mehr Sinn gemacht hat. Man hat dort die Beschleunigung wirklich gespürt. Unter korrekten Bedingungen scheinen die Partikel doch recht abrupt, aber nach Abklärung mit anderen Studenten, ist das wohl normal.

Abbildung 1

2. Grundlegende Analyse des Codes

Ganz allgemein ist das ein Beispiel der Nutzung des schon integrierten Beschleunigungssensor eines Android Gerätes. Hier als Beispiel einfache Kugeln die sich frei durch die Neigung des Gerätes bewegen.

Zum Code:

Public void onCreate(savedInstanceState)

- Wird aufgerufen, wenn die „Activity“ als erstes erstellt wird. Diese hol Instanzen von jeweils dem SensorManager, PowerManager, WindowManager, sowie erstellt einen WakeLock und initialisiert die Simulationsansicht, wo direkt der Hintergrund gezeichnet wird.

Protected void onResume ()

- Wenn die „Activity“ fortgesetzt wird, ein „WakeLock“ geholt, sodass der Bildschirm an bleibt.

Protected void onPause ()

- Wenn die „Activity“ pausiert ist, wird die Simulation gestoppt und die Sensorressourcen, sowie „WakeLocks“ freigesetzt.

Class SimulationView extends FrameLayout implements SensorEventListener {

- Klasse für die Simulationsansicht. Sie beinhaltet alle Parameter sowie die gesamte Logik bzw. Physik der Partikel.
- Alle Folgenden Methoden und Klassen sind innerhalb dieser Klasse

Class Particle extends View {

- Von jeder Partikel wird die momentane, vorherige Position gespeichert, sowie die Beschleunigung.

Public void computePhysics (float sx, float sy, float dT)

- Berechnung der Physik der Partikel.

Public void resolveCollisionWithBounds ()

- Behebung von Einschränkungen und Kollisionen zwischen Partikeln mittels des Verlet-Algorithmuses.

Class ParticleSystem {

- Das Partikelsystem ist nur eine Anhäufung von Partikeln. Hier wird festgelegt wie viele Partikel angezeigt werden. Standartgemäß auf 5 statisch festgelegt.

Private void updatePositions (float sx, float sy, long timestamp)

- Aktualisierung der Position jedes Partikels im Partikelsystem mittels des Verlet-Algorithmuses.

Public void update (float sx, float sy, long now)

- Vorerst eine Aktualisierung der Position aller Partikel. Anschließend Behandlung der Kollisionen zwischen den Partikeln. Es wird erst berechnet ob sich die Partikel berühren und wenn sie sich berühren, wird mathematisch eine Art von Abprall simuliert.

Public int getParticleCount ()

- Gibt Anzahl von Partikeln aus.

Public float getPosX (int i)

- Gibt die X-Position eines Partikels

Public float getPosY (int i)

- Gibt die Y-Position eines Partikels aus

Public Simulationview (Contex context)

- Sorgt für Korrekte anzeige der Simulation. Beispielsweise wird die Größe der Partikel für den Bildschirm angepasst.

Protected void onSizeChanged (int w, int h, int oldw, int oldh)

- Passt die Größe des Bildschirms relativ zu der Größe der Bitmap an.

Public void onSensorChanged (SensorEvent event)

- Hier wird die Rotation des Gerätes ausgelesen, um dann die entsprechenden Werte zu setzen.

Protected void onDraw (Canvas canvas)

- Tatsächliche Zeichnung aller Elemente

3. Anpassung des MQTT-Python Codes

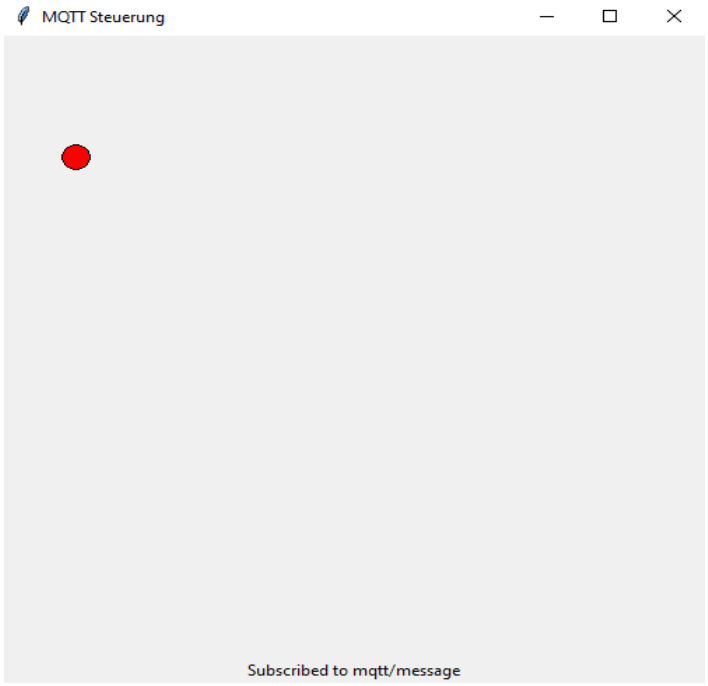

Abbildung 2

Anfangs habe ich jeglichen Code bezüglich des SenseHat entfernt und ein MQTT-Steuerelement wie in Abbildung 3, zu sehen, erstellt.

Die im Beispielcode enthaltene paint() Funktion habe ich in eine follow_mouse(cursor) Methode geändert, diese wie folgt definiert ist:

```
def follow_mouse( cursor ):
    x, y = cursor.x, cursor.y
    w.coords(char, x - 10, y - 10, x + 10, y + 10)
    #umwandlung der Koordinaten in einen -9.81 - 9.81 Intervall
    x = (x - (canvas_width/2)) / ((canvas_width/2)/9.81);
    y = (y -(canvas_height/2))/ ((canvas_width/2)/9.81);

    #publish an sensor/data der x und y Werte
    client.publish(pub_topic, str(x) + "," + str(y))
```

Abbildung 3

Die Erhaltenen Koordinaten werden direkt innerhalb dieser Funktion umgewandelt in Werte im Intervall von -9.81 – 9.81, um dann direkt die alten Werte des Accelerometers innerhalb der primären App zu ersetzen. Diese werden per client.publish() Methode innerhalb der follow_mouse Funktion direkt gepublisht.

4. Erweiterung der AccelerometerPlay-App um MQTT-Funktionalität

Die Funktionen connect(), subscribe(), publish() und disconnect() wurden ganz unten in der AccelerometerPlayActivity.java Datei definiert. Connect() sowie subscribe() in onResume() und disconnect() in onPause() initialisiert.

Anschließend wurde die onSensorChanged() angepasst. Dort werden nicht mehr die Rotationen abgefragt und darauffolgend die Werte gesetzt, sondern es werden nun die Werte aus der MQTT-Steuerung benutzt. Die Vorzeichen der X und Y Werte wurden ebenfalls angepasst, um die Steuerung für den Benutzer etwas natürlicher zu machen. Nun bewegen sich die Kugeln beispielsweise mit einer Bewegung nach links, auf dem MQTT-Steuerelement, ebenfalls nach links innerhalb der App.

5. Neue Features in App einfügen

a) Menüpunkt mit Texteingabedialog zum Setzen der IP-Adresse des MQTT-Brokers.

Hierzu wurde erst durch Rechtsklick auf den „res“ Ordner eine neue Android Resource File erzeugt, mit dem ResourceType von „menu“. Innerhalb des menu Ordners wurde eine „menu.xml“ Datei erstellt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:app="http://schemas.android.com/apk/res-auto"
3      xmlns:android="http://schemas.android.com/apk/res/android">
4      <item android:id="@+id/ipadress"
5          android:title="Broker IP-Adresse"
6          app:showAsAction="never" />
7
8  </menu>
```

Abbildung 4

In dieser ist nur ein Item, nämlich der Unterpunkt für den IP-Texteingabedialog, denn die Unteraufgabe für das an/aus Schalten von Ton ist entfallen.

Innerhalb der Java Datei wurden zwei Funktion hinzugefügt:

- **Public boolean onCreateOptionsMenu(Menu menu)**
- **Public boolean onOptionsItemSelected(MenuItem item)**

Die einerseits für das Menü selbst und andererseits für das OnClickEvent für jedes Element im Menü zuständig sind. (Hier nur ein Element)

Anschließend wurde ein Funktion **public void inputDialog()** erstellt, die für die Anzeige des InputDialogs zuständig ist. Diese ist realisiert durch einen angepassten AlertDialog, denn es gibt keine standartgemäßen InputDialog in Java. Diese wird innerhalb der onOptionsItemSelectet() Funktion aufgerufen.

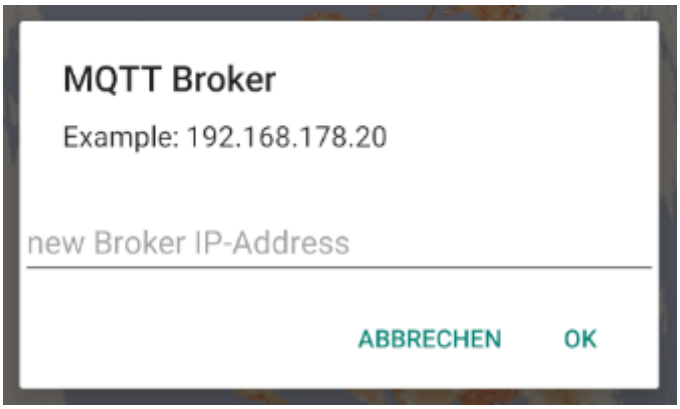

Abbildung 5

Die Eingabe ist so realisiert, dass der Benutzer nur die gewünschte IP-Adresse eingeben muss und nicht das Protokoll sowie den Port. Ich bin davon ausgegangen, dass das „tcp“ Protokoll und der „1883“ Port Standartgemäß gleichbleiben.

Nun zum Speicher der letzten IP-Adresse durch SharedPreferences:

Nach der globalen Definierung der nötigen Variablen, wurden eine loadData() Funktion definiert:

```
public void loadData() {
    SharedPreferences sharedPreferences = getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
    newIP = sharedPreferences.getString(NEW_IP,  defValue: "192.168.178.20");
}
```

Abbildung 6

Diese wird Anfangs in onCreate() aufgerufen und hat die IP „192.168.178.20“ als Standardwert festgelegt. Folglich wird der Input innerhalb der inputDialog() Funktion in den SharedPreferences gespeichert. Anschließend wird ein recreate() ausgeführt,

dass die neue IP-Adresse auch wirklich benutzt wird, denn loadData() wird wie schon erwähnt in onCreate() aufgerufen.

```
.setPositiveButton( text: "OK", (dialog, whichButton) → {
        //String inputIP = input.getText().toString();
        SharedPreferences sharedPreferences = getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString(NEW_IP, input.getText().toString());
        editor.apply();
        recreate();
})
```

Abbildung 7

b) Hintergrundbild und Kugeln-Bitmap austauschen

Die Eisenkugeln habe ich durch Virus-Zellen und der Holzhintergrund durch eine Weltkarte ersetzt.

Abbildung 8

c) Mit einer touch-Geste soll ein Kreis in den Bildschirm gezeichnet werden

Hierzu wurde anfangs ein setOnTouchListener() auf die mSimulationView in onCreate() gesetzt.

Anschließend eine **public boolean onTouch(View v, MotionEvent event)** Funktion erstellt, die bei einer onTouch-Aktion aufgerufen wird und an der Stelle die X und Y Koordinaten speichert. Ebenfalls wird ein boolean auf True gesetzt, welches eingebaut ist, um zu verhindert das direkt ein Kreis beim Starten der App gezeichnet wird. Abschließend wird in onDraw() der Kreis, an den gespeicherten Koordinaten mit einem festvorgegebenen Radius gezeichnet.

d) Entfernen der Kugeln bei Kollision mit dem eingezeichneten Kreis und anschließendes absetzten einer MQTT-Nachricht
e) Nicht bearbeitet

Die Erkennung, ob ein Partikel den eingezeichneten Kreis berührt ist mathematisch, in der onDraw() Funktion umgesetzt wurden. Erst wird der Abstand zwischen des Partikels und des Kreises über die Koordinaten errechnet und anschließend mit dem Gesamtradius verglichen. Wenn die Distanz <= gesamt Radius ist, berühren sich die Kreise. Anschließend wird ein removeView() ausgeführt, welches den kollidierten Partikel visuell entfernt. Daraufhin eine For-Schleife, die den Partikel aus dem Partikelarray entfernt. Abschließend wird der übrige Partikelcount per MQTT-Nachricht abgesetzt.

```
    //Radius in pixel (sBallDiameter ist in Metern)
    float particleR = (float)(mDstWidth/2);

    //Gesamtradius der beiden Kreise (Quadriert, sodass keine Wurzel benötig wird)
    float radiusSum = (((circleR) + (particleR)) * ((circleR) + (particleR)));
    //Distanz zwischen Kreisen errechnen. Auf Partikel Koordinaten den Radius drauf addieren, da der Mittelpunkt des Partikels oben links ist
    DistanceParticleCircle = (((x + particleR) - (circleX)) * ((x+ particleR) - (circleX))) + (((y+ particleR - circleY)) * ((y + particleR) - circleY));

    //Abfrage ob Distanz zwischen den Kreisen kleiner als der gemeinsame Radius der Kreise ist.
    if(DistanceParticleCircle <= radiusSum) {
        //Visuelle Entfernung der Partikel
        removeView(particleSystem.mBalls[i]);

        //logische Entfernung der Partikel
        for(int n = i; n < particleSystem.actualCount - 1; n++) {
            particleSystem.mBalls[n] = particleSystem.mBalls[n+1];
        }
        particleSystem.actualCount -= 1;
        publish( msg: "Partikelcount:" + String.valueOf(particleSystem.actualCount));
    }
}
```

Abbildung 9

Abschließend wurde der Code bereinigt. Unnötige Imports, Variablen und Funktionen wurden entfernt, sowie entsprechende Kommentare hinzugefügt.