MMI 2: Mobile Human-Computer Interaction

Übung 2

Prof. Dr. Michael Rohs

michael.rohs@ifi.lmu.de

Mobile Interaction Lab, LMU München

ORGANISATORISCHES

Exercises

#	Date	Topic
1	24.10.2011	Mobile usage scenarios
2	31.10.2011	Touch screen input
3	7.11.2011	Animations
4	14.11.2011	Exchanging data
5	21.11.2011	Location-based audio
6	28.11.2011	Paper-prototyping a mobile application
7	5.12.2011	Evaluating the paper prototype
8	12.12.2011	Visualizing off-screen data
9	19.12.2011	Interacting with small targets
10	9.1.2012	Tactile feedback
11	16.1.2012	Feature recognition
12	23.1.2012	Feature recognition
13	30.1.2012	Gesture recognition
14	6.2.2012	Exam preparation

Lectures

#	Date	Topic
1	19.10.2011	Introduction to Mobile Interaction, Mobile Device Platforms
2	26.10.2011	History of Mobile Interaction, Mobile Device Platforms
3	2.11.2011	Mobile Input and Output Technologies
4	9.11.2011	Mobile Interaction Design Process
5	16.11.2011	Mobile Communication
6	23.11.2011	Location and Context
7	30.11.2011	Prototyping Mobile Applications
8	7.12.2011	Evaluation of Mobile Applications
9	14.12.2011	Visualization and Interaction Techniques for Small Displays
10	21.12.2011	Mobile Devices and Interactive Surfaces
11	11.1.2012	Camera-Based Mobile Interaction 1
12	18.1.2012	Camera-Based Mobile Interaction 2
13	25.1.2012	Sensor-Based Mobile Interaction 1
14	1.2.2012	Sensor-Based Mobile Interaction 2
15	8.2.2012	Exam

ÜBUNG 2

Prof. Dr. Michael Rohs, Dipl.-Inform. Sven Kratz

Mensch-Maschine-Interaktion 2 (Mobile Interaktion) WS 2011/2012

Übungsblatt 2

Aufgabe 1: Android 2D Grafik, Touch Input [Einzelabgabe]

Schreiben Sie eine Anwendung, mit der der Benutzer mit dem Finger durch ein Labyrinth auf dem Display navigieren kann (siehe Beispiel-Screenshots unten). Die Aufgabe des Benutzers besteht darin, sich mit dem Finger ausgehend vom Start-Quadrat oben links zum Ziel-Quadrat unten rechts durch das Labyrinth zu bewegen, ohne den Finger vom Display abzuheben. Jede Berührung einer Labyrinth-Wand soll als Fehler gezählt und unten links angezeigt werden. Bei erfolgreicher Ausführung (keine Fehler) soll eine entsprechende Mitteilung auf dem Display erscheinen (z.B. als Meldung unten rechts).

Bei der Lösung der Aufgabe dürfen Sie ein Programmgerüst verwenden, das auf der Webseite der Vorlesung zu finden ist. Dieses kann einfach in Eclipse importiert werden. Das Labyrinth ist in einer XML-Datei spezifiziert, die in MainActivity.java eingelesen wird. Die Wände des Labyrinths liegen in einem Array (lines) vor.

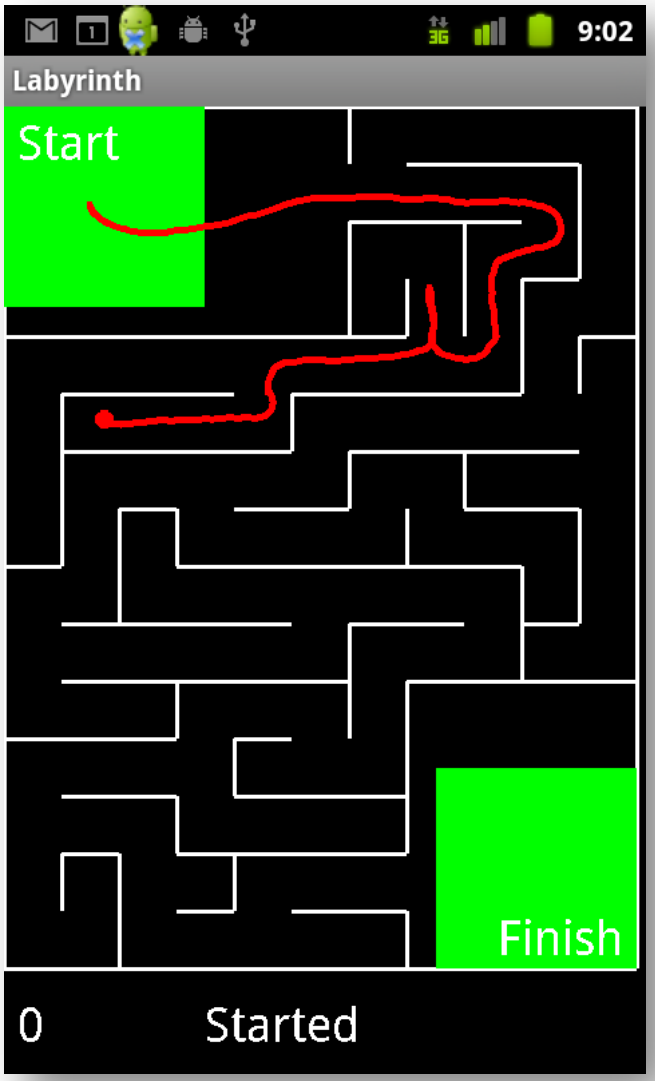
Da der Finger den Berührpunkt verdeckt, soll ein „Offset-Cursor“ verwendet werden. D.h. der Berührpunkt wird nach oben versetzt, so dass er nicht vom Finger verdeckt ist (siehe Abb. 1, rechts). Diese Idee wurde zuerst beschrieben in: Potter, Weldon, Shneiderman: Improving the accuracy of touch screens: an experimental evaluation of three strategies. CHI 1988.

Die Lösung der Aufgabe soll im Einzelnen wie folgt vorgenommen werden:

a) Importieren Sie das Programmgerüst in Eclipse.
b) Implementieren Sie ein Options-Menü (als Bestandteil der MainActivity), mit dem man zwischen 0 Offset und 100 Pixel vertikalem Offset umschalten kann.
c) Zeichnen Sie das Labyrinth in der Methode onDraw der Klasse LabyrinthView. Verwenden Sie die Linien-Daten aus der XML-Datei (gespeichert in Lines).
d) Zeichnen Sie die Start- und Zielfelder.
e) Fügen Sie Berührpunkte, die über onTouchEvent gemeldet werden zu einer Liste bzw. einem Array hinzu. Zeichnen Sie die eingegebene Spur. Bei erneuter Eingabe (MotionEvent.ACTION_DOWN) soll eine neue Spur gezeichnet werden.
f) Implementieren Sie die Methode countCollisions, die die Anzahl der Kollisionen der Eingabespur (bzw. der Strecke zwischen den beiden letzten Berührpunkten) mit den Labyrinth-Wänden berechnet. Beachten Sie die Kommentare im Programmgerüst.
g) Geben Sie die bisherige Anzahl an Kollisionen links unten auf dem Display aus. Optional: Falls Sie ein Gerät benutzen, geben Sie Vibrations-Feedback, falls eine Kollision erfolgt.
h) Geben Sie Feedback, falls das Labyrinth erfolgreich durchquert wurde.
i) Erstellen Sie ein zweites, einfacheres Labyrinth in XML, das ohne Offset-Cursor bedient werden kann. Wie groß sollten die sollten die Wege zwischen den Linien dabei gewählt werden (so dass auch ohne Offset-Cursor kaum Kollisionen vorkommen)?

Android 2D Grafik, Touch Input

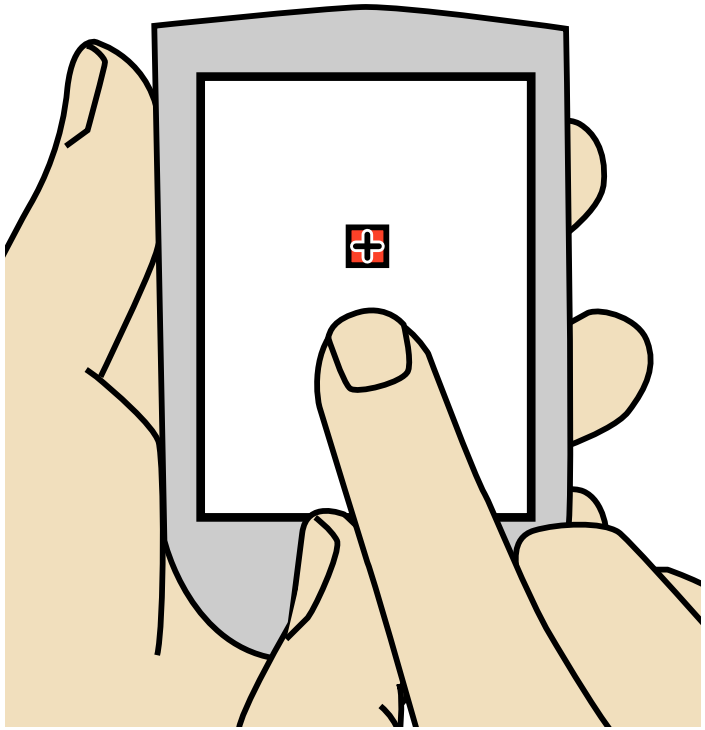
- Touch-Screen-Labyrinth
  - Finger von links oben nach rechts unten
  - Fehler bei Berührung von Wand
  - Anzahl Fehler unten links
  - Meldung, bei Erfolg (unten rechts oder pop-up)
- Programmgerüst → Webseite
- „Offset-Cursor“ da Finger Berührpunkt verdeckt



Offset-Cursor

- Berührpunkt wird nach oben versetzt, so dass nicht vom Finger verdeckt ist

- Potter, Weldon, Shneiderman: Improving the accuracy of touch screens: an experimental evaluation of three strategies. CHI 1988.



Labyrinth-Spezifikation /res/xml/labyrinth.xml

```
<?xml version="1.0" encoding="utf-8"?>
<labyrinth>
    <line x="6" y="0" length="1" orientation="S"/>
    <line x="7" y="1" length="3" orientation="E"/>
    <line x="10" y="1" length="2" orientation="S"/>
    <line x="6" y="2" length="3" orientation="E"/>
    <line x="6" y="2" length="2" orientation="S"/>
    <line x="8" y="2" length="2" orientation="S"/>
    <line x="9" y="3" length="1" orientation="E"/>
    …
</labyrinth>
```

x, y, length skalieren

S = south, E = east, …

Labyrinth-Spezifikation parsen

```
public Vector<Line> parseLabyrinthXml() throws IOException, XmlPullParserException
{
    XmlResourceParser parser = getResources().getXml(R.xml.labyrinth);
    Vector<Line> lines = new Vector<Line>();
    parser.next();
    int eventType = parser.getEventType();
    while (eventType != XmlPullParser.END_DOCUMENT) {
        switch (eventType) {
        case XmlPullParser.START_TAG:
            if ("line".equals(parser.getName())) {
                // nächste Folie
            }
            break;
            ...
            eventType = parser.next();
    }
    return lines;
}
```

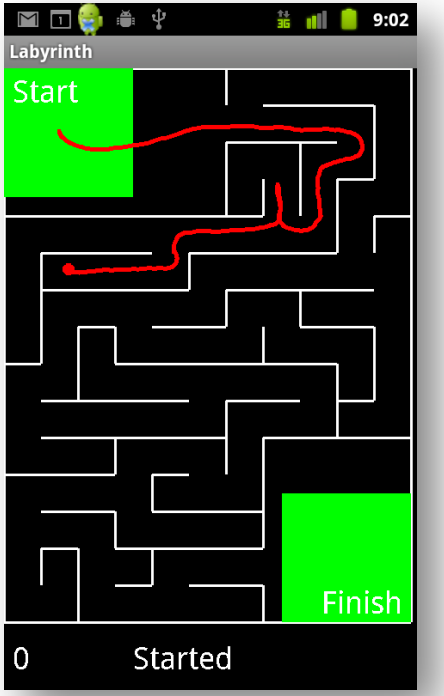
Labyrinth-Spezifikation parsen (cont‘d.)

```
Line line = new Line();
line.x1 = parser.getAttributeIntValue(null, "x", 0);
line.y1 = parser.getAttributeIntValue(null, "y", 0);
line.length = parser.getAttributeIntValue(null, "length", 0);
String o = parser.getAttributeValue(null, "orientation").toUpperCase();
if ("E".equals(o)) {
    line.orientation = Line.EAST;
    line.x2 = line.x1 + line.length;
    line.y2 = line.y1;
}
...
lines.add(line);
```

Line

```
public class Line {
        // line from point (x1,y1) to point (x2,y2)
        public int x1;
        public int y1;
        public int x2;
        public int y2;

        public int length;

        public int orientation; // 0 = E, 1 = N, 2 = W, 3 = S

        public static final int EAST = 0; // to the right
        public static final int NORTH = 1; // up
        public static final int WEST = 2; // to the left
        public static final int SOUTH = 3; // down

        public Line(…) { …}
}
```

Vorgehen (siehe auch Code-Kommentare)

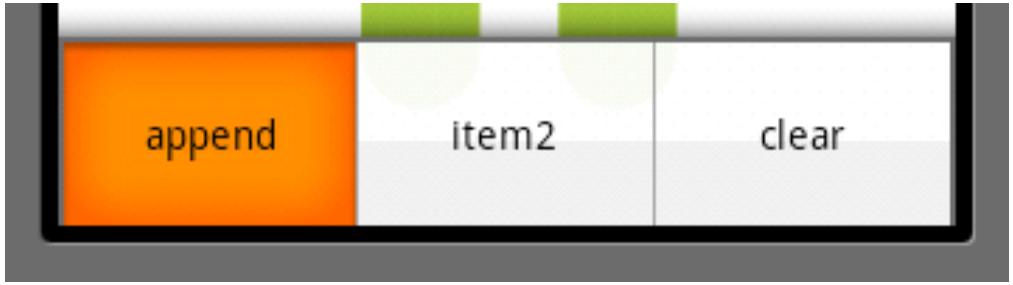


a) Programmgerüst importieren
b) Options-Menü (Offset 0 bzw. 100 Pixel)
c) Labyrinth zeichnen
d) Start- und Zielfelder zeichnen
e) onTouchEvent: MotionEvents in Liste, Spur zeichnen
f) countCollisions implementieren
g) Anzahl Kollisionen links unten ausgeben, optional: Vibrations-Feedback bei Kollision
h) Meldung, falls Labyrinth erfolgreich durchquert
i) (ganz) einfaches Labyrinth in XML erstellen, Bedienung ohne Offset-Cursor, Größe der Wege?

b) Options-Menü (Offset 0 bzw. 100 Pixel)

- An activity is associated with a single menu
- Use onCreateOptionsMenu(Menu m) to populate menu
- Creating an options menu

```
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, 1, 0, "append"); // group, id, order, title
    menu.add(0, 2, 1, "item2");
    menu.add(0, 3, 2, "clear");
    return true; // return true to enable menu
}
```



Responding to Menu Selection

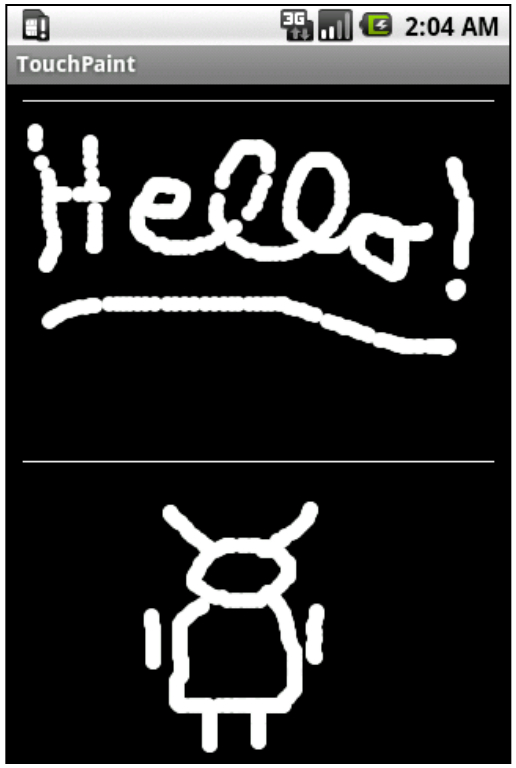
- Overriding onOptionsItemSelected

```
public boolean onOptionsItemSelected(MenuItem item) {
    Log.d("MainActivity", "menu id = " + item.getItemId() +
            ", title = " + item.getTitle().toString());
    switch (item.getItemId()) {
        case X: // id of handeled item
        // handle item X
        return true;
        ...
    }
}
```

c) Labyrinth zeichnen

relevante Methoden:
setARGB
setTextSize
setStrokeWidth
setStrokeCap
drawLine
drawRect

```
public class MyView extends View {
    private final Paint paint = new Paint();
    private int x = 0, y = 0;

    public MyView(Context c) {
        super(c);
        paint.setARGB(255, 255, 255, 255);
    }

    protected void onDraw(Canvas c) {
        c.drawCircle(x, y, 3, paint);
    }

    public boolean onTouchEvent(MotionEvent e) {
        x = (int)e.getX(); y = (int)e.getY();
        invalidate();
        return true;
}   }
```



e) onTouchEvent: MotionEvents in Liste
Touch Input: MotionEvent

- Method View.onTouchEvent(MotionEvent e)
- Motion event data
  - **x**, **y**, time, **action**, source, pressure, size
- Sources depend on hardware
  - Mouse, pen, finger, trackball
- Actions
  - ACTION_DOWN
  - ACTION_MOVE
  - ACTION_UP
  - ACTION_CANCEL
- Motion history
  - Sequence of coordinates between events

neue Spur anfangen

weitere Punkte zu aktueller Spur

Spur beenden

f) countCollisions implementieren – Ideen?

h) Meldung, falls erfolgreich durchquert Android Toast

- Package android.widget

```
import android.widget.Toast;
```

- Beispiel

```
String s = "Finished with " + collisions + " collisions!";
Toast toast = Toast.makeText(context, s, Toast.LENGTH_LONG);
toast.show();
```

context wird dem Konstructor von LabyrinthView übergeben

Hinweise

- Verwenden Sie keine Klassen des Packages android.gesture.
- Beachten Sie die Kommentare im Programmgerüst
- Das Programm muss kompilieren, sonst wird es nicht korrigiert!

Abgabe

- Plagiate sind verboten und führen zum Ausschluss aus der Veranstaltung!
  - Sie dazu auch die Hinweise zu Plagiaten. www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf

- Dieses Übungsblatt muss einzeln bearbeitet werden. Es darf nicht in Gruppen bearbeitet werden.
- Exportieren Sie Ihr Projekt aus Eclipse (Export → Archive file) und geben Sie es als zip-Datei bis zum 7.11.2011 um 12:00 Uhr im **neuen** UniWorX Portal (https://uniworx.ifi.lmu.de/) ab.

- Sie sollten Ihre Lösung in der Übung vorstellen können!