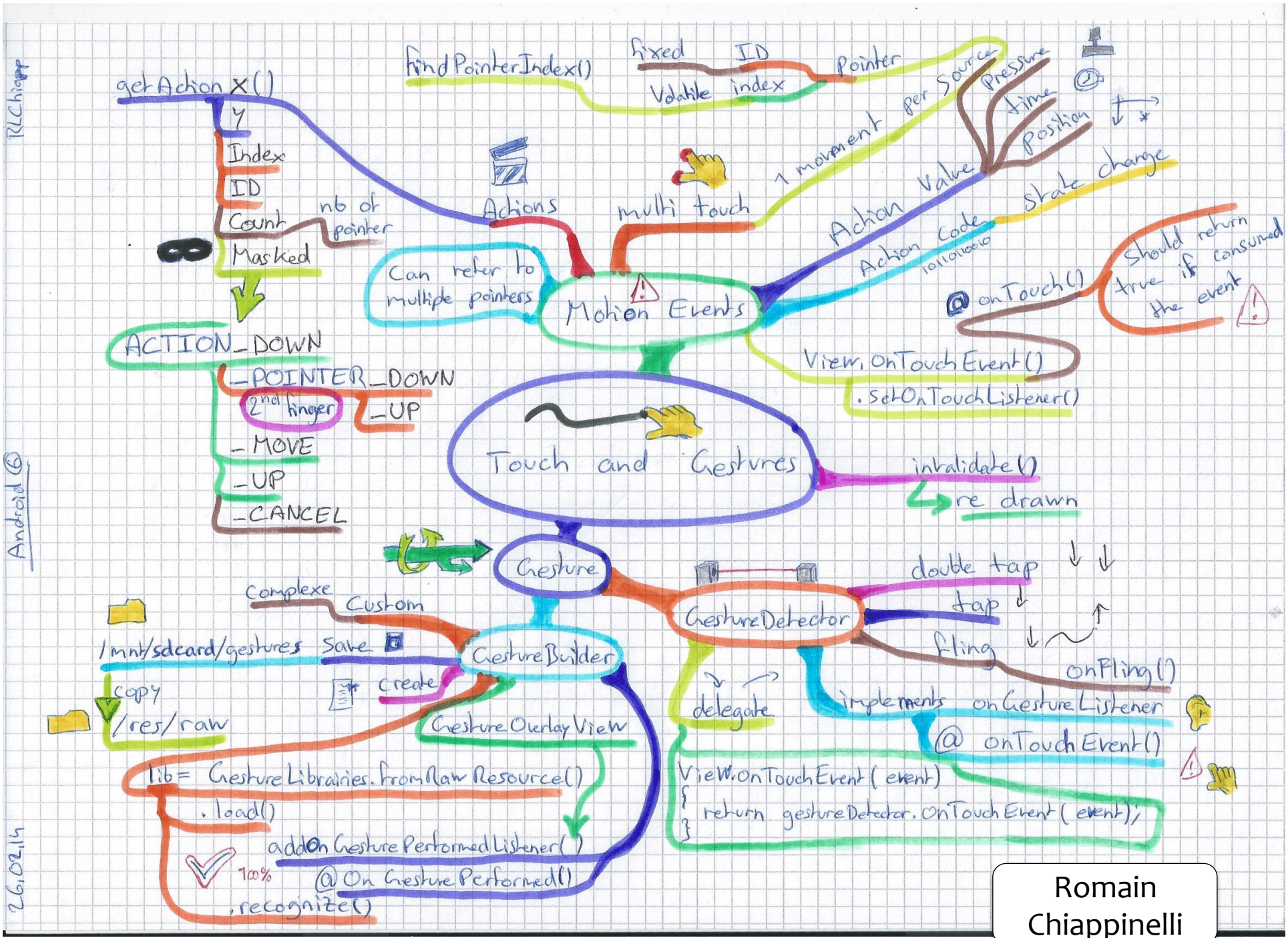

RLChiapp
Android ©
26.02.14
getAction X()
Y
Index
ID
Count
nb of pointer
Masked
ACTION_DOWN
_POINTER_DOWN
2nd finger
_UP
_MOVE
_UP
_CANCEL
findPointerIndex()
fixed ID
Volatile index
Pointer
Actions
multi touch
1 movement per Source
Pressure
time
Position
Action Value
Action Code
State change
Can refer to multiple pointers
Motion Events
@ onTouch()
Should return true if consumed the event
View.OnTouchEvent()
.setOnTouchListener()
Touch and Gestures
invalidate()
re drawn
Gesture
Complexe
Custom
/mnt/sdcard/gestures
Save
copy
/res/raw
GestureBuilder
create
GestureOverlayView
lib = GestureLibraries.fromRawResource()
.load()
addOnGesturePerformedListener()
@ On GesturePerformed()
100%
.recognize()
GestureDetector
double tap
tap
fling
onFling()
delegate
implements onGestureListener
@ onTouchEvent()
View.onTouchEvent(event)
{
return gestureDetector.onTouchEvent(event);
}
Romain Chiappinelli

# Multi-Touch & Gestures

# Today's Topics

MotionEvents

Touch Handling

Gestures

# MotionEvent

Represents a movement in an input device reading

pen, trackball, mouse, finger

# MotionEvent

## Action Code

State change that occurred

## Action Values

Position and movement properties, such as time, source, location, pressure, and more

This lesson focuses on touch events read from a touch screen

# MultiTouch

MultiTouch screens emit one movement trace per touch source

Individual touch sources are called pointers

# MultiTouch

Each pointer has a unique ID for as long as it is active

MotionEvents can refer to multiple pointers

Each pointer has an index within the event, but that index may not be stable over time

# Some MotionEvent actions

ACTION_DOWN

ACTION_POINTER_DOWN

ACTION_POINTER_UP

ACTION_MOVE

ACTION_UP

ACTION_CANCEL

# Consistency Guarantees

For touch events, Android tries to guarantee that touches

- Go down one at a time
- Move as a group
- Come up one at a time or are cancelled

Applications should be tolerant to inconsistency

# MotionEvent methods

getActionMasked()

getActionIndex()

getPointerId(int pointerIndex)

getPointerCount()

getX(int pointerIndex)

getY(int pointerIndex)

findPointerIndex (int pointerId)

# Handling Touch Events on a View

The View being touched receives
View.onTouchEvent(MotionEvent event)

onTouchEvent() should return true if the MotionEvent has been consumed; false otherwise

# Handling Touch Events with a Listener

View.OnTouchListener defines touch event callback methods

View.setOnTouchListener() registers listener for Touch callbacks

# Handling Touch Events with a Listener

onTouch() called when a touch event, such as pressing, releasing or dragging, occurs

onTouch() called before the event is delivered to the touched View

Should return true if it has consumed the event; false otherwise

# Handling Multiple Touch Events

multiple touches combined to form a more complex gesture

Identify & process combinations of touches,

For example, a double tap

ACTION_DOWN, ACTION_UP, ACTION_DOWN, ACTION_UP in quick succession

# Multi-touch Handling

# Multi-touch Handling

| | Action | IDs |
|---|---|---|
| #1 touch → | ACTION_DOWN | 0 |
| | ACTION_MOVE … | 0 |
| #2 touch → | ACTION_POINTER_DOWN | 1 |
| | ACTION_MOVE … | 0,1 |
| #1 lift → | ACTION_POINTER_UP | 0 |
| #2 lift → | ACTION_UP | 1 |

# Multi-touch Handling

# Multi-touch Handling

| | Action | ID |
|---|---|---|
| #1 touch → | ACTION_DOWN | 0 |
| | ACTION_MOVE … | 0 |
| #2 touch → | ACTION_POINTER_DOWN | 1 |
| | ACTION_MOVE … | 0,1 |
| #2 lift → | ACTION_POINTER_UP | 1 |
| #1 lift → | ACTION_UP | 0 |

# Multi-touch Handling

# Multi-touch Handling

| | Action | ID |
|---|---|---|
| #1 touch → | ACTION_DOWN | 0 |
| #2 touch → | ACTION_POINTER_DOWN | 1 |
| #3 touch → | ACTION_POINTER_DOWN | 2 |
| | ACTION_MOVE | 0,1,2 |
| #2 lift → | ACTION_POINTER_UP | 1 |
| #1 lift → | ACTION_POINTER_UP | 0 |
| #3 lift → | ACTION_UP | 2 |

# TouchIndicateTouchLocation

Application draws a circle wherever the users touches the screen

circle's color is randomly selected

Redraws circles when user drags finger across the screen

# TouchIndicateTouchLocation

The size of the circles are proportional to the number of currently active touches


18:31
TouchIndicateTouchLocation

# TouchIndicateTouchLocation

```
public class IndicateTouchLocationActivity extends Activity {

    private static final int MIN_DXDY = 2;

    // Assume no more than 20 simultaneous touches
    final private static int MAX_TOUCHES = 20;

    // Pool of MarkerViews
    final private static LinkedList<MarkerView> mInactiveMarkers = new LinkedList<MarkerView>();

    // Set of MarkerViews currently visible on the display
    @SuppressLint("UseSparseArrays")
    final private static Map<Integer, MarkerView> mActiveMarkers = new HashMap<Integer, MarkerView>();

    protected static final String TAG = "IndicateTouchLocationActivity";

    private FrameLayout mFrame;
```

# TouchIndicateTouchLocation

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mFrame = (FrameLayout) findViewById(R.id.frame);

    // Initialize pool of View.
    initViews();
```

# TouchIndicateTouchLocation

```
// Create and set on touch listener
mFrame.setOnTouchListener(new OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        switch (event.getActionMasked()) {

        // Show new MarkerView

        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_POINTER_DOWN: {

            int pointerIndex = event.getActionIndex();
            int pointerID = event.getPointerId(pointerIndex);

            MarkerView marker = mInactiveMarkers.remove();

            if (null != marker) {
                mActiveMarkers.put(pointerID, marker);

                marker.setXLoc(event.getX(pointerIndex));
                marker.setYLoc(event.getY(pointerIndex));

                updateTouches(mActiveMarkers.size());

                mFrame.addView(marker);
            }
            break;
        }
```

# TouchIndicateTouchLocation

```
// Remove one MarkerView

case MotionEvent.ACTION_UP:
case MotionEvent.ACTION_POINTER_UP: {

    int pointerIndex = event.getActionIndex();
    int pointerID = event.getPointerId(pointerIndex);

    MarkerView marker = mActiveMarkers.remove(pointerID);

    if (null != marker) {

        mInactiveMarkers.add(marker);

        updateTouches(mActiveMarkers.size());

        mFrame.removeView(marker);
    }
    break;
}
```

# TouchIndicateTouchLocation

```
// Move all currently active MarkerViews

case MotionEvent.ACTION_MOVE: {

    for (int idx = 0; idx < event.getPointerCount(); idx++) {

        int ID = event.getPointerId(idx);

        MarkerView marker = mActiveMarkers.get(ID);
        if (null != marker) {

            // Redraw only if finger has travel ed a minimum distance
            if (Math.abs(marker.getXLoc() - event.getX(idx)) > MIN_DXDY
                    || Math.abs(marker.getYLoc()
                            - event.getY(idx)) > MIN_DXDY) {

                // Set new location

                marker.setXLoc(event.getX(idx));
                marker.setYLoc(event.getY(idx));

                // Request re-draw
                marker.invalidate();
            }
        }
    }

    break;
}
```

# TouchIndicateTouchLocation

```
                default:

                    Log.i(TAG, "unhandled action");
                }

                return true;
            }

            // update number of touches on each active MarkerView
            private void updateTouches(int numActive) {
                for (MarkerView marker : mActiveMarkers.values()) {
                    marker.setTouches(numActive);
                }
            }
        });
    }

    private void initViews() {
        for (int idx = 0; idx < MAX_TOUCHES; idx++) {
            mInactiveMarkers.add(new MarkerView(this, -1, -1));
        }
    }
```

# TouchIndicateTouchLocation

```
private class MarkerView extends View {
    private float mX, mY;
    final static private int MAX_SIZE = 400;
    private int mTouches = 0;
    final private Paint mPaint = new Paint();

    public MarkerView(Context context, float x, float y) {
        super(context);
        mX = x;
        mY = y;
        mPaint.setStyle(Style.FILL);

        Random rnd = new Random();
        mPaint.setARGB(255, rnd.nextInt(256), rnd.nextInt(256),
                rnd.nextInt(256));
    }
```

# TouchIndicateTouchLocation

```
    float getXLoc() {
        return mX;
    }

    void setXLoc(float x) {
        mX = x;
    }

    float getYLoc() {
        return mY;
    }

    void setYLoc(float y) {
        mY = y;
    }

    void setTouches(int touches) {
        mTouches = touches;
    }

    @Override
    protected void onDraw(Canvas canvas) {
        canvas.drawCircle(mX, mY, MAX_SIZE / mTouches, mPaint);
    }
}
```

# GestureDetector

A class that recognizes common touch gestures

Some built-in gestures include confirmed single tap, double tap, fling

# GestureDetector

Activity creates a GestureDetector that implements the GestureDetector.OnGestureListener interface

Activity will receive calls to onTouchEvent() when activity is touched

onTouchEvent delegates call to GestureDetector.OnGestureListener

# TouchGestureViewFlipper

Shows a TextView displaying a number

If the user performs a right to left "fling" gesture,

The TextView will scroll off the screen

A new TextView will scroll in behind it


22:09
TouchGestureViewFlipper
1

# TouchGestureViewFlipper

```
public class ViewFlipperTestActivity extends Activity {
    private ViewFlipper mFlipper;
    private TextView mTextView1, mTextView2;
    private int mCurrentLayoutState, mCount;
    private GestureDetector mGestureDetector;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mCurrentLayoutState = 0;
```

# TouchGestureViewFlipper

```
        mFlipper = (ViewFlipper) findViewById(R.id.view_flipper);
        mTextView1 = (TextView) findViewById(R.id.textView1);
        mTextView2 = (TextView) findViewById(R.id.textView2);

        mTextView1.setText(String.valueOf(mCount));

        mGestureDetector = new GestureDetector(this,
                new GestureDetector.SimpleOnGestureListener() {
                    @Override
                    public boolean onFling(MotionEvent e1, MotionEvent e2,
                            float velocityX, float velocityY) {
                        if (velocityX < -10.0f) {
                            mCurrentLayoutState = mCurrentLayoutState == 0 ? 1
                                    : 0;
                            switchLayoutStateTo(mCurrentLayoutState);
                        }
                        return true;
                    }
                });
    }
```

# TouchGestureViewFlipper

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    return mGestureDetector.onTouchEvent(event);
}

public void switchLayoutStateTo(int switchTo) {
    mCurrentLayoutState = switchTo;

    mFlipper.setInAnimation(inFromRightAnimation());
    mFlipper.setOutAnimation(outToLeftAnimation());

    mCount++;

    if (switchTo == 0) {
        mTextView1.setText(String.valueOf(mCount));
    } else {
        mTextView2.setText(String.valueOf(mCount));
    }

    mFlipper.showPrevious();
}
```

# TouchGestureViewFlipper

```
private Animation inFromRightAnimation() {
    Animation inFromRight = new TranslateAnimation(
            Animation.RELATIVE_TO_PARENT, +1.0f,
            Animation.RELATIVE_TO_PARENT, 0.0f,
            Animation.RELATIVE_TO_PARENT, 0.0f,
            Animation.RELATIVE_TO_PARENT, 0.0f);
    inFromRight.setDuration(500);
    inFromRight.setInterpolator(new LinearInterpolator());
    return inFromRight;
}

private Animation outToLeftAnimation() {
    Animation outtoLeft = new TranslateAnimation(
            Animation.RELATIVE_TO_PARENT, 0.0f,
            Animation.RELATIVE_TO_PARENT, -1.0f,
            Animation.RELATIVE_TO_PARENT, 0.0f,
            Animation.RELATIVE_TO_PARENT, 0.0f);
    outtoLeft.setDuration(500);
    outtoLeft.setInterpolator(new LinearInterpolator());
    return outtoLeft;
}
```

# Creating Custom Gestures

The GestureBuilder application lets you create & save custom gestures

Comes bundled with SDK

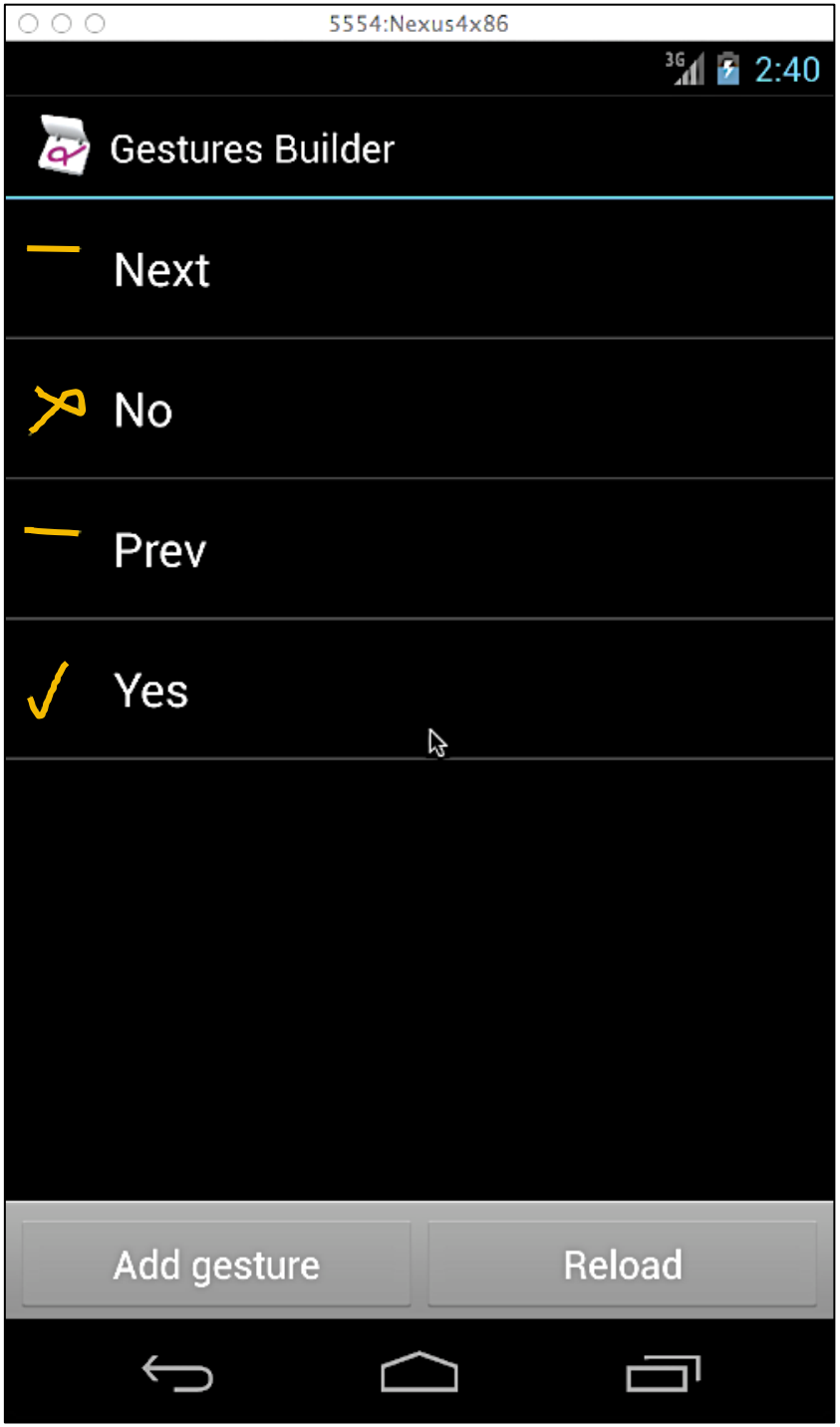
# Creating Custom Gestures

GestureLibraries supports loading custom gestures & then recognizing them at runtime

# Creating Custom Gestures

Include a GestureOverlayView in your layout

The Overlay intercepts user gestures and invokes your application code to handle them




# GestureBuilder

# GestureBuilder

Stores gestures to
/mnt/sdcard/gestures

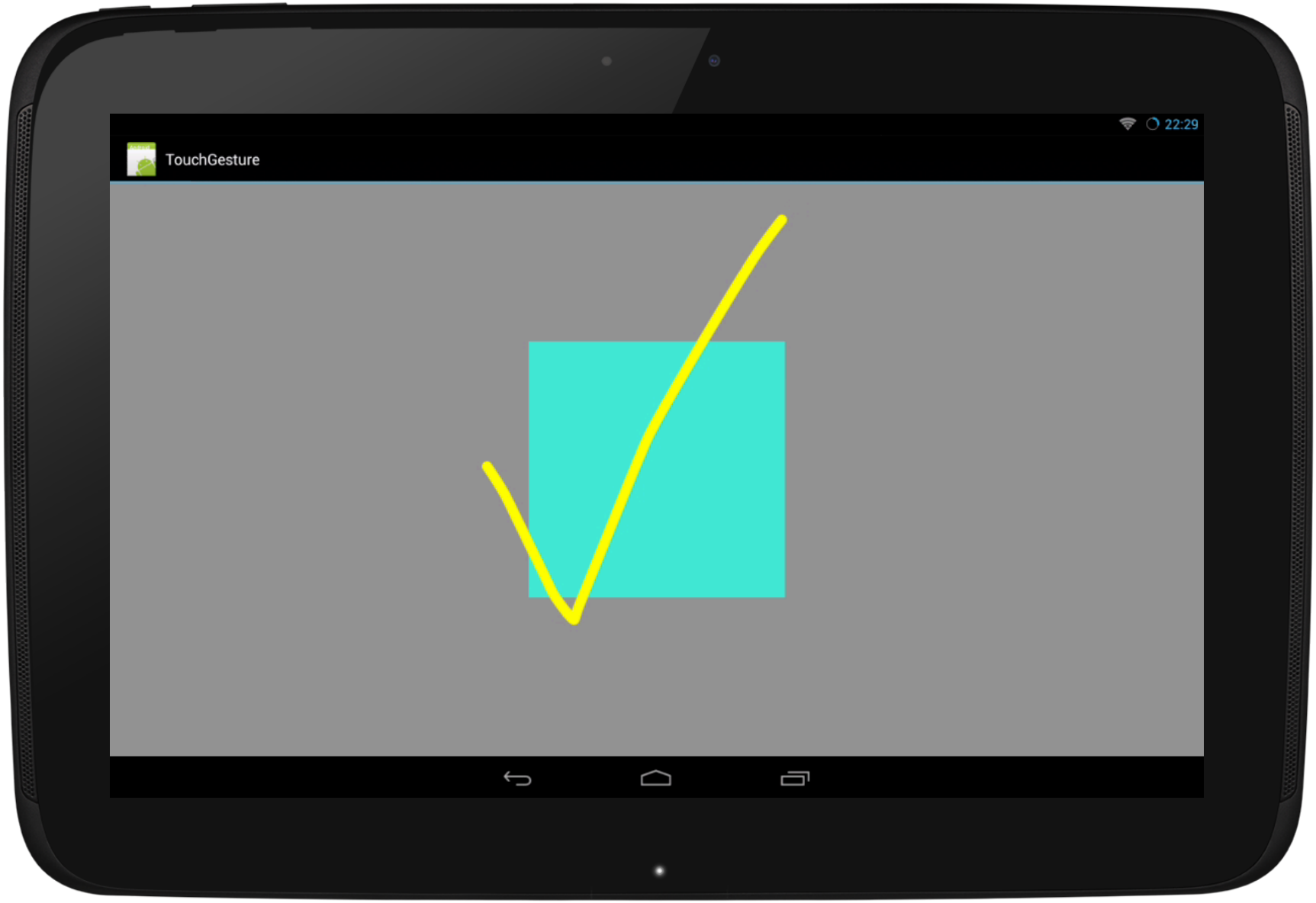
Copy this file to
/res/raw directory

# TouchGestures

Application displays a small View with a colored background

User can swipe left and right to cycle between different candidate background colors

Can make an check or X-like gesture to set or cancel the application's current background color


22:29
TouchGesture

# TouchGestures

```
public class GesturesActivity extends Activity implements
        OnGesturePerformedListener {
    private static final String NO = "No";
    private static final String YES = "Yes";
    private static final String PREV = "Prev";
    private static final String NEXT = "Next";
    private GestureLibrary mLibrary;
    private int mBgColor = 0;
    private int mFirstColor, mStartBgColor = Color.GRAY;
    private FrameLayout mFrame;
    private RelativeLayout mLayout;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

# TouchGestures

```
        mFrame = (FrameLayout) findViewById(R.id.frame);
        mBgColor = new Random().nextInt(0xFFFFFF) | 0xFF000000;
        mFirstColor = mBgColor;
        mFrame.setBackgroundColor(mBgColor);

        mLayout = (RelativeLayout) findViewById(R.id.main);
        mLayout.setBackgroundColor(mStartBgColor);

        mLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
        if (!mLibrary.load()) {
            finish();
        }

        // Make this the target of gesture detection callbacks
        GestureOverlayView gestureView = (GestureOverlayView) findViewById(R.id.gestures_overlay);
        gestureView.addOnGesturePerformedListener(this);

    }
```

# TouchGestures

```
public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture) {

    // Get gesture predictions
    ArrayList<Prediction> predictions = mLibrary.recognize(gesture);

    // Get highest-ranked prediction
    if (predictions.size() > 0) {
        Prediction prediction = predictions.get(0);

        // Ignore weak predictions

        if (prediction.score > 2.0) {
            if (prediction.name.equals(PREV)) {

                mBgColor -= 100;
                mFrame.setBackgroundColor(mBgColor);

            } else if (prediction.name.equals(NEXT)) {

                mBgColor += 100;
                mFrame.setBackgroundColor(mBgColor);
```

# TouchGestures

```
            } else if (prediction.name.equals(YES)) {

                mLayout.setBackgroundColor(mBgColor);

            } else if (prediction.name.equals(NO)) {

                mLayout.setBackgroundColor(mStartBgColor);
                mFrame.setBackgroundColor(mFirstColor);

            } else {
                Toast.makeText(this, prediction.name, Toast.LENGTH_SHORT)
                        .show();

            }
        } else {
            Toast.makeText(this, "No prediction", Toast.LENGTH_SHORT)
                    .show();
        }
    }
}
```

# Next Time

## MultiMedia