Распознавание пользовательских жестов[из песочницы](https://habrahabr.ru/sandbox/) tutorial

[Разработка под Android](https://habrahabr.ru/hub/android_dev/)\*

Недавно, при разработке игры под Android, я столкнулся с проблемой реализации работы с пользовательскими жестами. В стандартной комплектации Android SDK имеется класс GestureDetector*([тут](http://www.androidsnippets.com/gesturedetector-and-gesturedetectorongesturelistener) демонстрация работы с этим классом)*, однако в нём реализованы не все жесты, что мне были нужны, а также некоторые из них работали не так, как мне надо *(onLongPress, например, срабатывал не только по длительному касанию, но и по длительному касанию с ведением пальца по экрану)*. Кроме игр жесты могут использоваться и в обычных приложениях. Они могут заменить некоторые элементы интерфейса, тем самым сделав его проще. Жесты уже используются в очень многих приложениях для устройств с сенсорным вводом и это даёт нам право предполагать, что пользователь уже знаком с ними. Сегодня мы реализуем в нашем приложении распознавание [long press](http://developer.android.com/design/media/gesture_longtouch.png), [double touch](http://developer.android.com/design/media/gesture_doubletouch.png), [pinch open](http://developer.android.com/design/media/gesture_pinchopen.png), [pinch close](http://developer.android.com/design/media/gesture_pinchclose.png) и других.  
  


Препродакшн

Для примера работы с сенсорным экраном, multitouch и жестами я решил реализовать простенький графический редактор. Что должен представлять из себя наш графический редактор? Небольшой таскающийся холст, расстояние до которого можно менять, разводя и сводя пальцы. Также, чтобы не путать перетаскивание холста и рисование по нему, мы должны реализовать смену режима по длительному касанию. По двойному касанию можно отображать окошко выбора цвета.

Основа приложения

Для начала создаём Android Application Project в Eclipse. Наш проект не будет использовать каких-либо функций из новых SDK, поэтому в качестве Minimum Required SDK можно поставить API 8 *(Android 2.2)*, например. MainActivity *(если его нету, то нужно будет создать)* приведём к такому виду:

**import** android.os.Bundle;

**import** android.app.Activity;

**public** **class** **MainActivity** **extends** **Activity** {

**@Override**

**public** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(**new** ApplicationView(**this**));

}

}

Соответственно, мы должны создать класс ApplicationView, который будет наследовать от View:

**import** android.view.View;

**import** android.content.Context;

**import** android.graphics.Color;

**import** android.graphics.Paint;

**import** android.graphics.Canvas;

**public** **class** **ApplicationView** **extends** **View** {

Paint paint = **new** Paint(); *// Стиль рисования, кисть*

**static** **float** density;

**public** **ApplicationView**(Context context) {

**super**(context);

density = getResources().getDisplayMetrics().density;

**this**.setBackgroundColor(Color.GRAY); *// Устанавливаем серый цвет фона*

paint.setStrokeWidth(5\*density); *// Устанавливаем ширину кисти в 5dp*

}

**@Override**

**protected** **void** **onDraw**(Canvas canvas) {

}

}

Теперь нам надо разобраться, как мы будем отображать холст. Мы создаём Canvas, передав ему в параметры Bitmap. В данном случае при рисовании чего-либо на нашем Canvas, оно отображается на Bitmap. Так же нам нужна переменная позиции нашего холста и расстояния до него. Добавляем в класс ApplicationView несколько переменных:

Canvas canvas; *// Холст*

Bitmap image; *// Содержимое холста*

**float** zoom = 500; *// Расстояние до холста*

Point position = **new** Point(50, 50); *// Позиция холста*

В конструкторе класса инициализируем холст и его содержимое, а так же закрашиваем его белым цветом:

image = Bitmap.createBitmap(500, 500, Config.ARGB\_4444); *// Создаём содержимое холста*

canvas = **new** Canvas(image); *// Создаём холст*

canvas.drawColor(Color.WHITE); *// Закрашиваем его белым цветом*

В методе onDraw отображаем его:

canvas.translate(position.x, position.y); *// Перемещаем холст*

canvas.scale(zoom / 500, zoom / 500); *// Изменяем расстояние до холста*

canvas.drawBitmap(image, 0, 0, paint); *// Рисуем холст*

Если мы сейчас запустим наше приложение, мы увидим белый квадрат на сером фоне.

Сделать лучше

Реализация работы с сенсорным экраном

Нам нужно начать начать взаимодействовать с сенсорным экраном с помощью переопределения метода onTouchEvent *(подробнее о работе с сенсорным экраном*[*тут*](http://habrahabr.ru/post/111405/)*)*:

ArrayList<Finger> fingers = **new** ArrayList<Finger>(); *// Все пальцы, находящиеся на экране*

**@Override**

**public** **boolean** **onTouchEvent**(MotionEvent event) {

**int** id = event.getPointerId(event.getActionIndex()); *// Идентификатор пальца*

**int** action = event.getActionMasked(); *// Действие*

**if**(action == MotionEvent.ACTION\_DOWN || action == MotionEvent.ACTION\_POINTER\_DOWN)

fingers.add(event.getActionIndex(), **new** Finger(id, (**int**)event.getX(), (**int**)event.getY()));

**else** **if**(action == MotionEvent.ACTION\_UP || action == MotionEvent.ACTION\_POINTER\_UP)

fingers.remove(fingers.get(event.getActionIndex())); *// Удаляем палец, который был отпущен*

**else** **if**(action == MotionEvent.ACTION\_MOVE){

**for**(**int** n = 0; n < fingers.size(); n++){ *// Обновляем положение всех пальцев*

fingers.get(n).setNow((**int**)event.getX(n), (**int**)event.getY(n));

}

*//checkGestures();*

invalidate();

}

**return** **true**;

}

Мы видим, что при касании пальца, в список fingers мы добавляем объект типа Finger. Когда палец поднимаем, мы удаляем этот объект, а при перемещении пальцев мы обновляем координаты всех объектов. В итоге список fingers содержит в себе все касания с актуальными и предыдущими координатами. Класс Finger:

**import** android.graphics.Point;

**public** **class** **Finger** {

**public** **int** ID; *// Идентификатор пальца*

**public** Point Now;

**public** Point Before;

**boolean** enabled = **false**; *// Было ли уже сделано движение*

**public** **Finger**(**int** id, **int** x, **int** y){

ID = id;

Now = Before = **new** Point(x, y);

}

**public** **void** **setNow**(**int** x, **int** y){

**if**(!enabled){

enabled = **true**;

Now = Before = **new** Point(x, y);

}**else**{

Before = Now;

Now = **new** Point(x, y);

}

}

}

Проверка правильности

Перемещение холста

Для реализации перемещения холста нам нужно в методе onTouchEvent, в блоке перемещения объекта, вызывать метод checkGestures, который будет работать с касаниями. Вызов этого метода там уже есть, однако под комментарием. Раскомментируем его и пишем сам метод:

**public** **void** **checkGestures**(){

Finger point = fingers.get(0); *// Получаем нужный палец*

position.x += point.Now.x - point.Before.x; *// Изменяем координаты*

position.y += point.Now.y - point.Before.y;

}

Можно запустить и поводить пальцем и если всё сделано правильно, то холст должен таскаться за ним.

Изменение расстояния до холста

Для реализации данного жеста нужно разделить всё содержимое метода на multitouch *(когда пальцев более одного)* и обычное касание. Если это multitouch, то мы будем постоянно проверять нынешнее расстояние между двумя пальцами и прошлое. Изменим содержимое метода checkGestures:

Finger point = fingers.get(0);

**if**(fingers.size() > 1){ *// Multitouch*

*// Расстояние между пальцами сейчас (now) и раньше (before)*

**float** now = checkDistance(point.Now, fingers.get(1).Now);

**float** before = checkDistance(point.Before, fingers.get(1).Before);

**float** oldSize = zoom; *// Запоминаем старый размер картинки*

zoom = Math.max(now - before + zoom, density \* 25); *// Изменяем расстояние до холста*

position.x -= (zoom - oldSize) / 2; *// Изменяем положение картинки*

position.y -= (zoom - oldSize) / 2;

}**else**{ *// Обычное касание*

position.x += point.Now.x - point.Before.x;

position.y += point.Now.y - point.Before.y;

}

В этом участке кода был использован метод checkDistance, который нужно добавить в ApplicationView. Вот его код:

**static** **float** **checkDistance**(Point p1, Point p2){ *// Функция вычисления расстояния между двумя точками*

**return** FloatMath.sqrt((p1.x - p2.x)\*(p1.x - p2.x)+(p1.y - p2.y)\*(p1.y - p2.y));

}

Теперь при касании двух пальцев и сведении\разведении их будет изменятся расстояние до холста. Если у вас эмулятор, то ничего не получится.

Изменение режима

Нам нужно создать переменную, которая будет отвечать за режим. Я назвал её drawingMode типа boolean. Для реализации долгого касания нам придётся использовать метод, который будет вызываться через время. Тут есть несколько вариантов развития событий:

1. мы пишем наш код в методе onDraw;
2. мы пишем наш код в onTouchEvent;
3. мы создаём таймер и пишем код в него;
4. мы создаём Handler и с помощью метода postDelayed вызываем наш Runnable;

В методе onDraw, по моему мнению, должно выполняться только отображение графики. В onTouchEvent писать можно только с учётом того, что палец будет перемещаться, тем самым постоянно вызывая этот метод. Таймер работает постоянно и это немного напрягает, поэтому мы будем использовать четвёртый вариант. В классе ApplicationView мы создаём переменную handler типа Handler, а так же добавляем в блок «if(action == MotionEvent.ACTION\_DOWN || action == MotionEvent.ACTION\_POINTER\_DOWN)» метода onTouchEvent строку:

handler.postDelayed(longPress, 1000);

Теперь надо создать Runnable с именем longPress:

Runnable longPress = **new** Runnable() {

**public** **void** **run**() {

}

};

Теперь нам в классе Finger надо создать переменную wasDown типа long, которая будет содержать в себе время нажатия *(пригодится для двойного касания)*. В конструкторе этой переменной надо задать значение System.currentTimeMillis(). Ещё нам надо добавить переменную startPoint типа Point, которая будет содержать в себе стартовую позицию пальца. Она должна содержать в себе то значение, что было передано в конструктор, или при первом вызове setNow. Так же нам надо создать переменную enabledLongTouch типа boolean, отображающую пригодность данного касания для реализуемого нами события. Нам надо постоянно проверять, не отошёл ли палец слишком далеко от старта. Этот функционал можно реализовать в setNow. В итоге должно получиться примерно так:

**import** android.graphics.Point;

**public** **class** **Finger** {

**public** **int** ID;

**public** Point Now;

**public** Point Before;

**public** **long** wasDown;

**boolean** enabled = **false**;

**public** **boolean** enabledLongTouch = **true**;

Point startPoint;

**public** **Finger**(**int** id, **int** x, **int** y){

wasDown = System.currentTimeMillis();

ID = id;

Now = Before = startPoint = **new** Point(x, y);

}

**public** **void** **setNow**(**int** x, **int** y){

**if**(!enabled){

enabled = **true**;

Now = Before = startPoint = **new** Point(x, y);

}**else**{

Before = Now;

Now = **new** Point(x, y);

**if**(ApplicationView.checkDistance(Now, startPoint) > ApplicationView.density \* 25)

enabledLongTouch = **false**;

}

}

}

Теперь в методе run нашего Runnable мы можем проверять, длительное ли это касание:

**if**(fingers.size() > 0 && fingers.get(0).enabledLongTouch){

fingers.get(0).enabledLongTouch = **false**;

drawingMode = !drawingMode;

vibrator.vibrate(80);

}

Сделать лучше

Рисование

Сейчас мы реализуем рисование на холсте и изменение размера кисти. Для этого мы каждую часть метода checkGestures разделим ещё на две части: режим рисования и обычный режим. В режиме рисования при касании мы просто будем вести линию, а в режиме рисования при multitouch мы будем изменять размер кисти. Вот так станет выглядеть метод checkGestures:

Finger finger = fingers.get(0);

**if**(fingers.size() > 1){

**float** now = checkDistance(finger.Now, fingers.get(1).Now);

**float** before = checkDistance(finger.Before, fingers.get(1).Before);

**if**(!drawingMode){

**float** oldSize = zoom;

zoom = Math.max(now - before + zoom, density \* 25);

position.x -= (zoom - oldSize) / 2;

position.y -= (zoom - oldSize) / 2;

}**else** paint.setStrokeWidth(paint.getStrokeWidth() + (now - before) / 8);

}**else**{

**if**(!drawingMode){

position.x += finger.Now.x - finger.Before.x;

position.y += finger.Now.y - finger.Before.y;

}**else**{

**float** x1 = (finger.Before.x-position.x)\*500/zoom; *// Вычисляем координаты*

**float** x2 = (finger.Now.x-position.x)\*500/zoom; *// с учётом перемещения*

**float** y1 = (finger.Before.y-position.y)\*500/zoom; *// и расстояния*

**float** y2 = (finger.Now.y-position.y)\*500/zoom;

canvas.drawLine(x1, y1, x2, y2, paint); *// Рисуем линию*

canvas.drawCircle(x1, y1, paint.getStrokeWidth() / 2, paint); *// Сглаживаем концы*

canvas.drawCircle(x2, y2, paint.getStrokeWidth() / 2, paint);

cursor = finger.Now;

}

}

В последней строке я задаю значение некому cursor. Это переменная типа Point, содержащая координаты курсора. Курсор нужен лишь для того, чтобы ориентироваться в размере кисти. Для отображения курсора в методе onDraw добавляем:

**if**(drawingMode){

**int** old = paint.getColor(); *// Сохраняем старый цвет*

paint.setColor(Color.GRAY); *// Курсор будет серого цвета*

canvas.drawCircle((cursor.x-position.x)\*500/zoom, (cursor.y-position.y)\*500/zoom,

paint.getStrokeWidth() / 2, paint); *// Рисуем курсор в виде круга*

paint.setColor(old); *// Возвращаем старый цвет*

}

Теперь мы можем перемещать холст, приближать, отдалять его, переходить в режим рисования, рисовать, изменять размер кисти. Осталось лишь реализовать выбор цвета.

Выбор цвета

Выбор цвета происходит после двойного касания по экрану. Для этого в ApplicationView нужно создать переменную, сохраняющую прошлое касание по экрану и переменную, сохраняющую координаты этого касания. Первая пусть будет называться lastTapTime типа long, а вторая — lastTapPosition типа Point. Тогда изменим метод onTouchEvent:

**int** id = event.getPointerId(event.getActionIndex());

**int** action = event.getActionMasked();

**if**(action == MotionEvent.ACTION\_DOWN || action == MotionEvent.ACTION\_POINTER\_DOWN)

fingers.add(event.getActionIndex(), **new** Finger(id, (**int**)event.getX(), (**int**)event.getY()));

**else** **if**(action == MotionEvent.ACTION\_UP || action == MotionEvent.ACTION\_POINTER\_UP){

Finger finger = fingers.get(event.getActionIndex()); *// Получаем нужный палец*

*// Если касание длилось менее 100мс, после предыдущего касания прошло не более 200мс,*

*// если касание было после предыдущего и дистанция между двумя касаниями не более 25dp*

**if**(System.currentTimeMillis() - finger.wasDown < 100 && finger.wasDown - lastTapTime < 200 &&

finger.wasDown - lastTapTime > 0 && checkDistance(finger.Now, lastTapPosition) < density \* 25){

*// Тут произошло двойное касание*

}

lastTapTime = System.currentTimeMillis(); *// Сохраняем время последнего касания*

lastTapPosition = finger.Now; *// Сохраняем позицию последнего касания*

fingers.remove(fingers.get(event.getActionIndex()));

}**else** **if**(action == MotionEvent.ACTION\_MOVE){

**for**(**int** n = 0; n < fingers.size(); n++){

fingers.get(n).setNow((**int**)event.getX(n), (**int**)event.getY(n));

}

checkGestures();

}

**return** **true**;

Нам осталось лишь реализовать диалог выбора цвета. Там, где происходит касание *(помечено комментарием)*, пишем:

Builder builder = **new** AlertDialog.Builder(getContext());

String[] items = {"Красный", "Зелёный", "Синий", "Голубой", "Чёрный", "Белый", "Жёлый", "Розовый"};

**final** AlertDialog dialog = builder.setTitle("Выберите цвет кисти").setItems(items, **new** DialogInterface.OnClickListener() {

**public** **void** **onClick**(DialogInterface dialog, **int** which) {

**int**[] colors = {Color.RED, Color.GREEN, Color.BLUE, 0xFF99CCFF, Color.BLACK, Color.WHITE,

Color.YELLOW, 0xFFFFCC99};

paint.setColor(colors[which]);

}

}).create();

dialog.show();

Если вы запустите приложение, то увидите что по двойному касанию появляется окошко выбора цвета.

Заключение

Распознавание пользовательских жестов оказалось не такой уж и сложной задачей. Мы разобрали эту на примере реализации графического редактора. Похожей реализацией, естественно, могут обладать не только приложения, но и игры.   
  
Исходники проекта [тут](https://github.com/suVrik/GesturesHabrahabr).  
  
Использовал информацию с [developer.android.com](http://developer.android.com/intl/ru/index.html). На началах использовал [эту](http://habrahabr.ru/post/111405/) статью.  
Так же хотел бы выразить благодарность пользователям [AndreyMI](https://habrahabr.ru/users/andreymi/), [LeoCcoder](https://habrahabr.ru/users/leoccoder/), [silentnuke](https://habrahabr.ru/users/silentnuke/) и