Поддержка устройств ввода – РАЗРАБОТКА ИГР ДЛЯ ОС ANDROID

Добавлено [Vadik](http://src-code.net/author/vadik/) дата October 2, 2013

[**0**](http://src-code.net/podderzhka-ustrojstv-vvoda-razrabotka-igr-dlya-os-android/#respond)

Как обсуждалось в Android мы получаем информацию от различных устройств ввода. В этом разделе мы обсудим три наиболее важных метода ввода и работу с ними: сенсорный экран, клавиатуру и акселерометр.

**Обработка событий касания и множественных касаний**

Сенсорный экран – вероятно, самый важный способ получения данных от пользователя. До версий Android 2.0 API поддерживал обработку только одиночных касаний. Мультитач был представлен в Android 2.0 (версия SDK 5). Его обработка была встроена в API для одиночных касаний, что дало неоднозначные результаты с точки зрения удобства. Для начала мы изучим обработку одиночных касаний, доступную во всех версиях Android.

Обработка одиночных касаний. Когда мы обрабатывали нажатия кнопки, то видели, что интерфейсы слушателя являются тем инструментом, с помощью которого Android сообщает нам о наступлении событий. То же самое и с касаниями – эти события передаются реализации интерфейса OnTouchLi stener, зарегистрированного нами с Vi ew. Интерфейс OnTouchLi stener включает в себя лишь один метод:

http://src-code.net/img/Android_Games_coding_image099.jpg

Первый аргумент – объект Vi ew, к которому относятся события касания. Второй аргумент – это то, что нам необходимо разобрать для получения этого события.

OnTouchLi stener может быть зарегистрирован в любой реализации Vi ew с помощью метода View. setOnTouchLi stenerC) и будет вызываться перед тем, как MotionEvent будет отправлен к View. Мы можем сообщить View в нашей реализации метода onTouch , что событие касания уже обработано, возвращая из метода true. При возврате значения f al se объект Vi ew сам будет обрабатывать это событие. Экземпляр MotionEvent обладает несколькими методами, нам интересны три из них.

MotionEvent.getX и MotionEvent.getY – возвращают координаты х и у точки касания внутри View. Как вы уже знаете, координаты определяются от левого верхнего угла – слева направо по оси х и сверху вниз по оси у. Значения координат возвращаются в пикселах. При этом тип возвращаемых этими методами значений – f 1oat, поэтому координаты имеют субпиксельную точность.

Moti onEvent. getActi on  – возвращает тип события касания. Это целочисленное значение, соответствующее одному из элементов списка Moti onEvent. ACTI0N D0WN, MotionEvent.ACTI0N M0VE, MotionEvent.ACTION CANCEL и MotionEvent.ACTIONJJP.

Звучит довольно просто, и это впечатление не обманчиво. Событие Moti onEvent. ACTI0ND0WN возникает, когда палец касается экрана. При движении пальцем по дисплею возникают события Moti onEvent. ACTI0N M0VE. Обратите внимание – вы всегда будете получать события Moti onEvent. ACTI0N M0VE, поскольку не сможете держать палец неподвижно, чтобы их избежать.

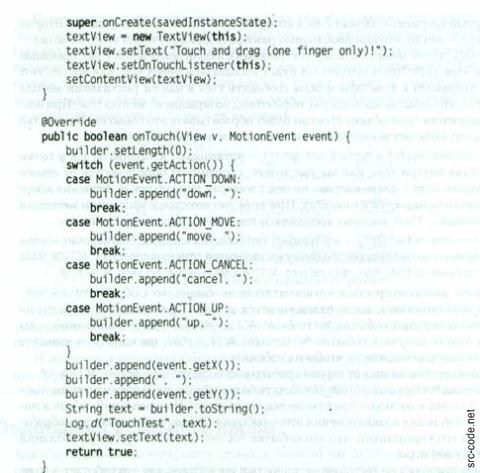
При поднятии пальца от экрана срабатывает событие Moti onEvent. ACTIONJJP.

События Moti onEvent. ACTION CANCEL чуть более загадочны. В документации говорится, что они возникают при отмене текущего жеста. Но в реальной работе я никогда не наблюдал возникновения этого события. Тем не менее мы будем обрабатывать и его (представив, что это событие Moti onEvent. ACTIONJJP) при создании нашей первой игры.

Создадим простую тестовую активность и посмотрим, как это работает в коде. Активность должна показывать текущую позицию пальца на дисплее, а также тип наступившего события. Листинг 4.3 демонстрирует, что я имею в виду.

Листинг 4.3. SingleTouchTest.java; тестирование обработки одиночных касаний package com.badogi с.androi dgames;

[](http://src-code.net/img/Android_Games_coding_image100.jpg)

[](http://src-code.net/img/Android_Games_coding_image101.jpg)

Наша активность реализует интерфейс OnTouchLi stener. В ней также есть два члена, один для TextVi ew, второй – StringBui 1 der, который мы будем использовать для создания строк для событий.

Метод onCreate  не требует длинных пояснений. Единственное новшество – вызов TextView.setOnTouchListener, в котором мы регистрируем нашу активность в TextVi ew, чтобы она могла получать MotionEvents.

Все, что осталось, – собственно реализация метода onTouchC). Мы игнорируем аргумент типа View, потому что знаем, что это наш объект TextVi ew. Нас интересует получение типа события, добавление его к объекту Stri ngBui 1 der, получение координат и обновление текста TextVi ew. Вот и все. Кроме того, мы записываем сообщения в LogCat, чтобы видеть порядок появления событий – ведь в TextVi ew мы увидим только последнее обработанное событие (StringBuider очищается при каждом вызове onTouchC).

Последняя небольшая подробность о методе onTouch – выражение return, в котором мы возвращаем true. Обычно мы придерживаемся концепции слушателя и возвращаем fal se, чтобы не связываться лишний раз с процессом обработки событий. Если мы сделаем так же в нашем примере, то не увидим никаких событий, кроме Moti onEvent. ACTI0N D0WN. Поэтому мы сообщаем TextVi ew, что мы только что использовали событие. В разных реализациях View поведение может различаться. К счастью, в нашей е будут применяться еще всего три вида View, и мы сможем использовать любое событие по нашему желанию.

Если мы запустим это приложение на эмуляторе или подключенном устройстве, то увидим, что TextVi ew всегда показывает тип последнего события и координаты, вычисленные в методе onTouch. Кроме того, все эти сообщения вы можете увидеть в LogCat.

Я не определял явно ориентацию активности в файле манифеста. Если вы повернете устройство так, чтобы активность перешла в ландшафтный режим, то координатная система, конечно, изменится. Рисунок 4.6 показывает активность в портретном и ландшафтном режимах. В обоих случаях я постарался касаться экрана в середине. Обратите внимание – координаты х и у как будто поменялись местами. На рисунок также добавлены оси и точка касания. В обоих случаях координаты начинаются в левом верхнем углу TextVi ew, ось х движется вправо, ось у – вниз.

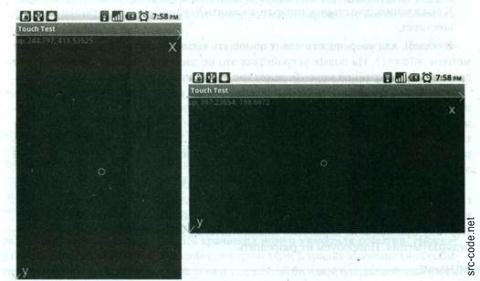
[](http://src-code.net/img/Android_Games_coding_image102.jpg)

Рис. 4.6. Касание экрана в портретном (слева) и ландшафтном (справа) режимах

В зависимости от ориентации меняются максимально возможные значения х и у. Для примера мы использовали Nexus One (разрешение 480 х 800 пикселов в портретном режиме и 800 х 480 в ландшафтном). Поскольку координаты касания даны относительно View (которое занимает не весь экран устройства), максимальное значение координаты х будет всегда меньше разрешения по высоте. Позже вы узнаете, как включить полноэкранный режим, чтобы убрать строку состояния и заголовок.

К сожалению, существуют некоторые проблемы с событиями касания на старых версиях Android и устройствах первого поколения.

Лавина событий касания. Драйвер сообщает о максимально возможном количестве событий, когда палец находится на экране, – на некоторых устройствах оно измеряется сотнями за секунду. Мы можем частично справиться с этой проблемой, поместив вызов Thread. sleep(16) в метод onTouchC) – после каждого события поток обработки будет приостанавливаться на 16 миллисекунд. В этом случае мы получим максимально 60 событий в секунду, чего более чем достаточно для нормального игрового процесса. Эта проблема касается только устройств на базе Android 1.5.

Касание экрана загружает процессор. Даже включив паузу в методе onTouch, мы должны понимать, что система обрабатывает процессы в своем ядре. На старых устройствах (например, Него или G1) эти действия могут захватывать до 50 % мощности процессора, что оставляет меньше пространства для потока главного цикла. Как следствие, падает частота кадров – иногда до такой степени, когда игровой процесс становится невозможным. На устройствах второго поколения проблема встречается намного реже, и о ней можно забыть. К сожалению, для старых аппаратов какого-то универсального решения не существует.

В общем, для уверенности стоит помещать вызов Thread.siеер(16) во все ваши методы onTouchO. На новых устройствах это не даст никакого эффекта; на более древних это по крайней мере убережет вас от лавины сообщений о касаниях.

Учитывая, что аппараты первого поколения все же постепенно уходят с рынка, проблема становится менее значимой с течением времени. Однако она по-прежнему может огорчать разработчиков игр. Попытайтесь оправдаться перед пользователями тем, что ваша игра работает как черепаха потому, что какой-то драйвер расходует все ресурсы процессора. Результат вполне предсказуем – пользователей это не будет волновать.

Обработка множественных касаний. Это были цветочки – пришло время главной головной боли. API для работы с мультитач было встроено в класс MotionEvent, изначально предназначенный для обработки отдельных касаний. Поэтому при попытке обработки множественных касаний возникают некоторые недоразумения. Попробуем их разрешить.

ПРИМЕЧАНИЕ

API для мультитач, похоже, вызывает некоторые вопросы даже у инженеров Android, создавших его. Он был сильно модернизирован в SDK версии 8 (Android 2.2) – новые методы, новые и даже переименованные константы. Эти изменения должны сделать работу с множественными касаниями немного проще, но доступны они только начиная с SDK версии 8. Для поддержки всех версий Android с мультитач (начиная с 2.0) придется использовать SDK версии 5.

Обработка множественных касаний довольно сильно похожа на обработку событий однократного касания. Мы реализуем тот же интерфейс OnTouchLi stener, что и для однократных касаний. Нам также необходим экземпляр MotionEvent, из которого будут считываться данные. Мы будем обрабатывать те же события, что и раньше (например, Moti onEvent. ACTIONJJP), и добавим к ним еще несколько.

Указатели ID и индексы. Различия начинаются, когда нам необходим доступ ккоординатам касания. MotionEvent. getX и Moti onEvent. getY возвращают координаты одного пальца на экране. При обработке событий множественных касаний нам необходимо использовать перегруженные версии этих методов для получения так называемого номера указателя. Это должно выглядеть так:

http://src-code.net/img/Android_Games_coding_image103.jpg

Теперь мы можем ожидать, что poi nterlndex напрямую связан с одним из пальцев, касающихся экрана (например, первый палец на экране получает poi nterlndex, равный 0, следующий – 1 и т. д.). К сожалению, это не так.

poi nterlndex – номер во внутренних массивах Moti onEvent, хранящих координаты события для определенного пальца, который касается дисплея. Реальный идентификатор пальца называется идентификатором указателя. Существует отдельный метод MotionEvent.getPointerldentifier(int pointerlndex), возвращающий идентификатор указателя, базирующийся на номере указателя. Идентификатор указателя для пальца не изменится, пока тот не оторвется от экрана (но это не обязательно верно для номера указателя).

Рассмотрим, как мы можем получить номер указателя для события. Пока проигнорируем тип события.

[http://src-code.net/img/Android_Games_coding_image104-min-vs.jpg](http://src-code.net/img/Android_Games_coding_image104.jpg)

Вы, видимо, подумали сейчас то же, что подумал я, когда впервые написал этот код. Однако прежде, чем окончательно потерять веру в человечество, попробуем расшифровать, что тут происходит. Мы получаем тип события из Moti onEvent через MotionEvent .getAction. Отлично, это мы уже освоили. Далее мы выполняем битовую операцию AND, используя полученное от метода Moti onEvent. getActionO значение и константу Moti onEvent. Теперь начинается самое веселое.

Константа имеет значение OxffOO, поэтому мы, по существу, делаем все биты равными 0, кроме битов с 8 по 15, хранящих номер указателя события. Нижние 8 бит числа, возвращенного методом event. get Act i on , хранят значение типа события (например, MotionEvent.ACTION JD0WN и т. д.). Этой битовой операцией мы, проще говоря, стираем данные о типе события.

Теперь сдвиг приобретает больший смысл. Мы осуществляем его с помощью константы MotionEvent. ACTION POIIMTER I D SH I FT (равной 8), то есть перемещаем биты с 8 по 15 в биты с 0 по 7, получая актуальный номер указателя на событие. Обратите внимание – наши волшебные константы называются XXX POINTER ID XXX, а не XXX POINTER INDEX XXX (что имело бы больше смысла – ведь нам необходим номер указателя, а не его идентификатор). Что ж, инженеры Android тоже ошибаются. В SDK версии 8 они убрали эти константы и представили взамен новые, названные XXX POINTER INDEX XXX и имеющие те же значения, что и убранные. Для обеспечения работы старых приложений, написанных на SDK 5, старые константы тоже сохранены.

Итак, мы знаем, как получить этот загадочный номер указателя, с которым мы можем запросить координаты и идентификатор указателя события.

Маска операции и другие типы событий. Далее нам необходимо получить настоящий тип события минус дополнительный номер указателя, закодированный в числе, возвращенном MotionEvent .getActionC). Нам лишь необходимо исключить номер указателя:

http://src-code.net/img/Android_Games_coding_image105.jpg

Да, это было просто. Увы, вы поймете это, только если знаете, что это номер указателя и что он зашифрован в действии.

Осталось декодировать тип события, как мы делали это ранее. Я уже упоминал, что у нас появились новые типы событий, поэтому рассмотрим их.

MotionEvent. ACT 10N P01 NTER D0WN – возникает для каждого пальца (начиная со второго), касающегося экрана. Первый палец все еще вызывает событие Moti onEvent. ACTI0N D0WN.

MotionEvent. ACT 10N P01NTE R U P – аналог предыдущего действия. Событие возникает, когда палец отрывается от экрана и. при этом экрана касаются более одного пальца. Последний палец при отрыве вызывает событие Moti onEvent.ACTIONJJP. Это не обязательно будет первый палец, коснувшийся экрана.

К счастью, мы можем представить, что два этих новых типа событий ничем не отличаются от старых добрых Moti onEvent. ACTIONJJP и Moti onEvent. ACTI0N D0WN.

Последнее отличие – тот факт, что один MotionEvent может обладать данными для множества событий. Да, вы все верно прочитали. По этой причине соединенные вместе события должны иметь одинаковый тип. В реальности это происходит только для события MotionEvent. ACTION J10VE, и нам приходится мириться с этим фактом. Чтобы проверить, сколько событий содержится в одном Moti onEvent, мы используем метод Moti onEvent. getPointerCountO, сообщающий нам о количестве пальцев, касающихся нашего дисплея. Далее мы можем получить идентификатор указателя и координаты для номеров указателя от О до MotionEvent.getPointerCountO –  с помощью методов Moti onEvent. getX, MotionEvent. getY и Moti onEvent. getPointerlcK).

**На практике**

Напишем пример для данного API. Нам хотелось отслеживать в нем касания всех 10 пальцев (на данный момент не существует устройств, способных отслеживать большее их количество, так что нам ничего не грозит). Android присвоит этим пальцам идентификаторы указателей от 0 до 9 в том порядке, в котором они касаются дисплея. Итак, мы будем хранить координаты каждого идентификатора и его состояние (касается или нет), а также выводить эту информацию на экран с помощью TextVi ew. Назовем нашу тестовую активность MultiTouchTest. Полный код показан в листинге 4.4.

Листинг 4.4. MultiTouchTest.Java; тестируем Multitouch API package com.bad ogi с.androi dgames;

[](http://src-code.net/img/Android_Games_coding_image106.jpg)

[](http://src-code.net/img/Android_Games_coding_image107.jpg)

Мы реализовали интерфейс OnTouchListener, как и раньше. Для отслеживания координат и состояния касания 10 пальцев мы добавили три члена-массива, в которых будет храниться нужная нам информация. Массивы хиу хранят координаты каждого ID указателя, массив touched содержит данные о том, касается ли данный ID экрана или нет.

Далее я создал небольшой вспомогательный метод, выводящий в TextVi ew текущее состояние пальцев. Он просто пробегает по всем 10 состояниям пальцев и склеивает их в одну строку с помощью Stri ngBui 1 der. Конечный текст выводится в TextView.

Метод onCreate устанавливает нашу активность и регистрирует ее в качестве OnTouchLi stener в TextVi ew. Эта часть кода вам уже должна быть знакома.

Теперь самая тяжелая часть – метод onTouchO. Начинаем с получения типа события на основе целого числа, возвращенного методом event. getActi on . Далее получаем номер указателя и соответствующий ему идентификатор из Moti onEvent, как уже обсуждалось ранее. Ядро метода onTouch – большое путаное выражение switch, которое мы уже использовали в усеченном виде при обработке одиночных касаний. Мы группируем события в три высокоуровневые категории.

Произошло касание экрана (MotionEvent.ACTI0N D0WN, MotionEvent.ACTI0N P0NTER  DOWN). Для таких идентификаторов указателей мы устанавливаем состояние касания в true, а также сохраняем текущие координаты этого указателя.

Произошел отрыв указателя от экрана (MotionEvent.ACTIONJJP, MotionEvent. ACTI0N P0INTER UP, Moti onEvent. CANCEL). Состояние касания для таких идентификаторов устанавливается в false, сохраняются их последние известные координаты.

Один или несколько пальцев двигаются по экрану (MotionEvent.ACTI0N M0VE). Мы проверяем количество событий, находящихся в данный момент в Moti onEvent, и затем обновляем координаты для номеров указателей от 0 до MotionEvent. getPointerCountO – 1. Для каждого события мы получаем соответствующий идентификатор указателя и обновляем его координаты.

При выполнении события мы обновляем TextVi ew с помощью вызова метода updateView, определенного нами ранее. Наконец, возвращаем значение true – знак того, что мы обработали событие касания.

Рисунок 4.7 демонстрирует вывод активности после того, как я коснулся экрана моего Nexus One двумя пальцами, а потом немного подвигал ими.

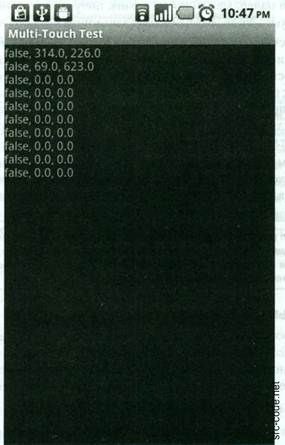


Рис. 4.7. Результат использования мультитач

Осталось несколько вопросов, которые стоит обсудить при запуске этого примера.

Если мы запустим его на эмуляторе с версией Android ниже 2.0, то получим ужасное исключение, поскольку мы используем отсутствующее в старых версиях API. С этим можно бороться, определяя в процессе работы приложения текущую версию Android и применяя в случае с Android 1.5 и 1.6 API для обработки одиночных касаний, а в случае с Android 2.0 и позже – мультитач API.

На эмуляторе нет поддержки мультитач. Мы можем создать его с использованием версии Android 2.0 и выше, но это ничего не даст – мышь-то у нас всего одна. И даже если бы их было две, это ничего бы не изменило.

Коснитесь двумя пальцами дисплея, поднимите первый и снова коснитесь им экрана. Второй палец сохранит свой идентификатор касания после того, как первый палец поднимается. Когда первый палец вновь касается дисплея, он получает первый свободный идентификатор указателя (в данном случае 0). Каждый новый палец, касающийся экрана, получает первый незанятый идентификатор. Это правило, которое стоит запомнить.

Если вы протестируете данный пример на Nexus One или Droid, то можете обратить внимание на несколько странное поведение при пересечении двух пальцев на одной оси. Это происходит из-за того, что экраны данных устройств не поддерживают в полной мере отслеживание отдельных пальцев. Это на самом деле большая проблема, но мы с ней справимся, разрабатывая наш интерфейс с некоторой оглядкой.

Пока просто держите это в голове.

Итак, вот таким образом осуществляется обработка множественных касаний на Android. Сложно, конечно, но, разобравшись один раз и примирившись с некоторой неуклюжестью реализации, этот функционал вполне можно использовать.

ПРИМЕЧАНИЕ

Простите, что взорвал вам мозг. Этот пункт и был действительно трудным. К сожалению, официальная документация для этого API невероятно лаконична, и большинство людей изучают эти возможности, кромсая API на практике. Я предлагаю вам поработать с вышеприведенным кодом,    пока вы не будете понимать точно все, что в нем делается.

Источник: Mario Zechner / Марио Цехнер, «Программирование игр под Android», пер. Егор Сидорович, Евгений Зазноба, Издательство «Питер»