**Статическое отображение картинок**

Основы игр будем изучать с помощью клубка. Для начала нам понадобится изображение клубка.

Клубок

Первый урок будет самым простым - научимся выводить клубок в центре экрана.

Для игр нам понадобится игровая площадка. Стандартные макеты **LinearLayout**, **RelativeLayout** и т.д. нам не подойдут, они разработаны для других целей и не предполагают активного взаимодействия, за исключением некоторых анимационных эффектов. Оставим их в покое.

Воспользуемся специальным классом **View**, который является базовым классом для всех стандартных компонентов, а также для создания собственных компонентов с нуля. У поля есть поверхность для рисования, называемое **Canvas** (Холст) и специальные методы для рисования объектов. Наследуемый от класса **View** наш новый класс послужит игровой площадкой для будущих игр.

Создадим новый класс **Board**, наследуясь от **View**:

public class Board extends View {

}

Студия предложит создать хотя бы один из четырёх возможных конструкторов. Выберем самый простой, остальные нам не понадобятся.

public Board(Context context) {

super(context);

}

У конструктора один параметр - контекст. Заготовка игровой площадки готова и мы можем вывести её на экран. Удаляем у активности метод подключения разметки, а вместо неё подключаем свой компонент.

// setContentView(R.layout.activity\_main);

// Подключаем вместо разметки

setContentView(new Board(this));

Если вы хотите видеть экран игры в полном размере без лишних заголовков, то до вызова метода **setContentView()** разместите следующий код:

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

getWindow().setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN,

WindowManager.LayoutParams.FLAG\_FULLSCREEN);

Или можете выбрать готовую тему для активности, которая прячет заголовки.

Я не буду этого делать, чтобы выводить информацию в заголовке программы. А вы можете использовать этот код.

Запустим проект и увидим пустой белый экран. Пока ничего интересного. А где же обещанный клубок? Продолжим работу с игровой площадкой **Board()**. Но сначала поместим подготовленное изображение клубка ниток в папку ресурсов **res/drawable**. Для простого примера я не буду подготавливать разные размеры клубка под разные разрешения экрана, нам важно сфокусироваться на других вещах.

Изображение клубка является растровой картинкой, состоящей из множества цветных точек. Для работы с такими картинками существует класс **Bitmap**. Объявим новую переменную в начале нашего класса для будущего объекта данного типа. А также объявим две переменные, которые будут отвечать за местоположение клубка на экране.

private Bitmap mBallBitmap;

В конструкторе класса мы создадим объект для нашего клубка, сообщив, что данные возьмём из ресурсов и сообщим идентификатор картинки в ресурсах:

mBallBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.klubok);

Итак, в памяти мы создали объект клубка, но экране мы его не увидим, потому что ещё не дали команды отобразить (нарисовать) его. Для рисования у класса **View** есть специальный метод **onDraw()** с единственным параметром **Canvas**, который предоставляет поверхность для рисования. Мы сообщаем, что хотим отобразить картинку - для этого есть специальный метод **drawBitmap()**. Также есть отдельные методы для рисования точки, линии, прямоугольника и даже слов (это как рисовать на заборе). Нам нужно указать объект рисования, его координаты и цвет для рисования.

canvas.drawBitmap(mBallBitmap, canvas.getWidth() / 2, canvas.getHeight() / 2,

null);

В нашем случае мы указали созданный ранее объект **mBallBitmap**, а координаты центра игровой площадки вычислили самостоятельно, разделив ширину и высоту доступной нам площадки пополам. Мы не привязываемся к конкретным значениям, поэтому на любых устройствах клубок будет находиться в центре экрана. Последний параметр нам не нужен, так как мы не собираемся пока ничего рисовать.

Впрочем, если приглядеться, то можно заметить, что клубок находится не совсем по центру. Кругом нас обманывают. Не кипятитесь. Дело в том, что отсчёт координат идёт не от центра клубка, а от левого верхнего угла квадрата, в который вписан наш клубок. Если вам действительно хочется увидеть клубок точно в центре, то вспоминайте математику. Нам придётся дополнительно вычислить ширину и высоту самого клубка и отнять половинки этих значений.

Целиком код для класса **Board** станет следующим:

package ru.alexanderklimov.ball;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.view.View;

public class Board extends View {

private Bitmap mBallBitmap;

public Board(Context context) {

super(context);

mBallBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.klubok);

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

canvas.drawBitmap(mBallBitmap, canvas.getWidth() / 2 - mBallBitmap.getWidth() / 2,

canvas.getHeight() / 2 - mBallBitmap.getHeight() / 2, null);

}

}

В данном примере мы получали размеры холста и вычисляли центр. Также можно получить размер самого компонента **View**. Добавим в класс новые поля и переопределим метод **on SizeChanged()**. И в методе **onDraw()** вычислим центр.

private int mScreenWidth;

private int mScreenHeight;

@Override

protected void onSizeChanged(int w, int h, int oldw, int oldh) {

super.onSizeChanged(w, h, oldw, oldh);

mScreenWidth = w;

mScreenHeight = h;

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

canvas.drawBitmap(mBallBitmap,

(mScreenWidth - mBallBitmap.getWidth()) / 2,

(mScreenHeight - mBallBitmap.getWidth()) / 2, null);

}

Какой из этих способов предпочтительнее, решать вам. В нашем примере разницы нет, но второй способ может пригодиться при использовании других методов, где нет доступа к объекту**Canvas**. Поэтому пока оставляем первый способ, а второй держим в уме. Метод **onSizeChanged()** вызывается после конструктора, но перед рисованием.

**Используем фигуры**

В примере мы использовали готовую картинку, которая загружается из ресурсов. Но графическим объектом может быть и фигура, нарисованная программно. Например, с помощью метода **canvas.drawCirlce()** можно нарисовать круг. Конечно, он будет не такой красивый, как клубок, но для некоторых игр подойдёт и такой вариант. Также можно нарисовать квадрат, треугольник, линию и т.д. Создадим похожий файл (или можете добавить в этот же проект). Для интерактивности добавим метод, который будет отслеживать касания экрана.

package ru.alexanderklimov.ball;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.view.MotionEvent;

import android.view.View;

public class Board extends View {

private Bitmap mBallBitmap;

// Для красного и зеленого круга

private Paint mPaint;

private int mRedCircleX, mGreenCircleX;

private int mRedCircleY, mGreenCircleY;

private float mRadius;

public Board(Context context) {

super(context);

mBallBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.klubok);

// Для красного и зеленого круга

mPaint = new Paint();

mPaint.setAntiAlias(true);

mRedCircleX = 20;

mRedCircleY = 20;

mRadius = 50;

mGreenCircleX = 400;

mGreenCircleY = 100;

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

canvas.drawBitmap(mBallBitmap, canvas.getWidth() / 2 - mBallBitmap.getWidth() / 2,

canvas.getHeight() / 2 - mBallBitmap.getHeight() / 2, null);

// Для красного круга

mPaint.setColor(Color.RED);

canvas.drawCircle(mRedCircleX, mRedCircleY, mRadius, mPaint);

// Для зеленого круга

mPaint.setColor(Color.GREEN);

canvas.drawCircle(mGreenCircleX, mGreenCircleY, mRadius, mPaint);

}

public boolean onTouchEvent(MotionEvent event) {

int eventAction = event.getAction();

int x = (int) event.getX();

int y = (int) event.getY();

switch (eventAction) {

case MotionEvent.ACTION\_DOWN:

break;

case MotionEvent.ACTION\_MOVE:

break;

case MotionEvent.ACTION\_UP:

mRedCircleX = x;

mRedCircleY = y;

break;

}

invalidate();

return true;

}

}

О работе с касаниями поговорим в другой раз. Пока только поясню, что в момент отрыва пальца (или лапы кота) от экрана, в методе **onTouchEvent()** мы получаем координаты и перемещаем красный круг в эту позицию.

