**Падают, падают, падают листья (ValueAnimator)**

Напишем пример с падающими листьями. Найдите подходящие картинки, например, эти:

Для реализма листья должны:

* падать вниз
* вращаться вокруг своей оси на заданный угол (покачиваться)
* падать не строго вертикально, а вдоль кривой (ветер)

Таким образом нам нужно реализовать анимацию перемещения и вращения.

Изображения листьев следует поместить в контейнеры **ImageView**, которые будут создаваться в случайном порядке и добавляться в текущую разметку активности перед запуска анимации. Для этого нам понадобится таймер, управляющий анимацией. Допустим, мы хотим видеть падающий лист каждые пять секунд. Нам понадобится объект **Handler**, который возьмёт на себя всю работу по анимации.

private Handler mHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

super.handleMessage(msg);

int viewId = new Random().nextInt(LEAVES.length);

Drawable d = getResources().getDrawable(LEAVES[viewId]);

LayoutInflater inflate = LayoutInflater.from(FallAnimationActivity.this);

ImageView imageView = (ImageView) inflate.inflate(R.layout.ani\_image\_view, null);

imageView.setImageDrawable(d);

mRootLayout.addView(imageView);

mAllImageViews.add(imageView);

LayoutParams animationLayout = (LayoutParams) imageView.getLayoutParams();

animationLayout.setMargins(0, (int) (-150 \* mScale), 0, 0);

animationLayout.width = (int) (60 \* mScale);

animationLayout.height = (int) (60 \* mScale);

startAnimation(imageView);

}

};

Объект **ImageView** создаётся из XML-файла. Также вычисляется размер экрана (mScale), чтобы определить позицию листа на 150 единиц выше видимой верхней части экрана.

Далее создаётся таймер.

private class ExeTimerTask extends TimerTask {

@Override

public void run() {

// we don't really use the message 'what' but we have to specify

// something.

mHandler.sendEmptyMessage(Constants.EMPTY\_MESSAGE\_WHAT);

}

}

Таймер запускается при запуске приложения.

new Timer().schedule(new ExeTimerTask(), 0, 5000);

Таким образом, мы создаём **ImageView** каждые пять секунд, добавляем его в родительскую разметку и запускаем анимацию.

Теперь займёмся самой анимацией. Задержка перед запуском анимации вычисляется случайным образом в диапазоне от 0 до 6000 миллисекунд. Сама анимация будет длится 10 секунд, можете поиграть с этим значением самостоятельно.

Координата x вычисляется случайным образом от 0 (левый край экрана) до значения ширины экрана (правый край экрана). Координата y должна находиться в диапазоне высоты экрана плюс 150dp.

Анимация вращения должна создавать иллюзию движения листа то в одну, то в другую сторону. Угол вращения от 50 до 100 вполне достаточен для эффекта.

Min + (int)(Math.random() \* ((Max - Min) + 1))

50 + (int)(Math.random() \* 101)

Анимация вращения и перемещения происходит в методе **startAnimation()**:

public void startAnimation(final ImageView aniView) {

aniView.setPivotX(aniView.getWidth() / 2);

aniView.setPivotY(aniView.getHeight() / 2);

long delay = new Random().nextInt(Constants.MAX\_DELAY);

final ValueAnimator animator = ValueAnimator.ofFloat(0, 1);

animator.setDuration(Constants.ANIM\_DURATION);

animator.setInterpolator(new AccelerateInterpolator());

animator.setStartDelay(delay);

animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {

int angle = 50 + (int) (Math.random() \* 101);

int movex = new Random().nextInt(mDisplaySize.right);

@Override

public void onAnimationUpdate(ValueAnimator animation) {

float value = ((Float) (animation.getAnimatedValue()))

.floatValue();

aniView.setRotation(angle \* value);

aniView.setTranslationX((movex - 40) \* value);

aniView.setTranslationY((mDisplaySize.bottom + (150 \* mScale))\* value);

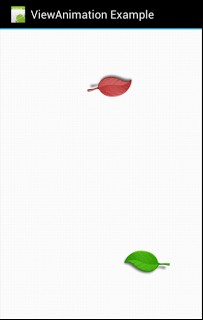
}

});

animator.start();

}

Анимация падающего листа представлена на видео.



На основе статьи [Android: Leaf fall-like animation using property animators | Java Code Geeks](http://www.javacodegeeks.com/2012/10/android-leaf-fall-like-animation-using.html).

Исходники: https://github.com/2dwarfs/FallAnimationSample